UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE GRADO

SISTEMA BASADO EN FPGA PARA LA CAPTURA DE TRÁFICO EN REDES MULTIGIGABIT ETHERNET

Autor: José Fernando Zazo Rollón,
Tutor: Dr. Sergio López Buedo

JULIO 2014

SISTEMA BASADO EN FPGA PARA LA CAPTURA DE TRÁFICO EN REDES MULTIGIGABIT ETHERNET

AUTOR: José Fernando Zazo Rollón
TUTOR: Dr. Sergio López Buedo

High Performance Computing and Networking Research Group
Dpto. de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
JULIO 2014

Resumen

***Resumen* —** Este trabajo consiste en el desarrollo de un sistema para la medición y captura de tráfico en redes de ordenadores multigabit ethernet, concretamente sobre enlaces a 10 Gigabits ($10 \cdot 10^9$ bits) por segundo. Sus aplicaciones son variadas y entre ellas se encuentran la monitorización de enlaces, observación de aplicaciones, evaluación de prestaciones, planificación de redes o análisis de seguridad, entre otras.

Se parte de la premisa de que existen soluciones comerciales capaces de operar en enlaces a 10 Gbps. Sin embargo, el principal problema localizado es el equipo necesario para la realización de la actividad. Típicamente se trata de estaciones de trabajo de una gran capacidad de cómputo y coste. En un equilibrio por buscar un prototipo de captura en redes multigigabit ethernet económico y funcional se implementa un sistema basado en dispositivos hardware programables (FPGAs), concretamente en el proyecto NetFPGA, por un total inferior a los 3100€.

Con dicha finalidad será necesario el desarrollo de un módulo propio del sistema operativo, intercambio de datos a través del bus PCI-Express del sistema usando la lógica de DMA (ofrecida por el core de Northwest Logic) y la creación de un dispositivo de almacenado que garantice la tasa multigigabit, un RAID de nivel 0 con 8 discos de estado sólido. De esta manera, con un diseño basado en FPGA, se eliminan los cuellos de botella del sistema (salvo por la gestión del RAID) y el procesamiento de la pila TCP/IP. El acceso a la memoria es paliado utilizando páginas de memoria de tamaño no convencional que permitirán la transferencia de regiones mayores en cada operación de DMA.

Como elemento diferenciador que impulse la aplicación más allá de la cuestión puramente monetaria, se destaca la interceptación de los datos al nivel de la capa física, en el protocolo XGMII. Las posibles incertidumbres asociadas a la temporización desaparecen al ser la propia aplicación la encargada de gestionar el protocolo, conociendo el momento exacto de llegada de cada trama. Ligado a la decisión de capturar en el nivel físico, se permite recibir paquetes con errores de CRC a nivel ethernet que de otra manera serían desechados.

Se concluye presentando los resultados teóricos esperables frente a los resultados reales, analizando las posibles diferencias entre ambos, destacando principalmente el rendimiento del RAID donde la escalabilidad no ha resultado lineal. Varios puntos sobre trabajo futuro a partir del diseño funcional de captura a 10 Gbps realizado son expuestos entre los que cabe destacar el desarrollo de un capturador a 40 Gbps partiendo del modelo presentado.

***Palabras Clave* —** 10g, Captura a 10g, Volcado a disco, FPGA, NetFPGA, PCIe, Huge pages, SSD RAID.

Abstract

***Abstract* —** This project involves the development of a system for measuring and capturing traffic in multigabit ethernet computer networks, specifically on 10 Gigabit ($10 \cdot 10^9$ bits) per second networks. Its applications are rich and includes network and application monitoring, performance evaluation, network planning or security analysis, amongst others.

The starting point is the premise that there are commercial solutions which can operate at 10 Gbps links. However, the main problem is located on the necessary computers for carrying out the activity. Typically, these workstations have a great computing capacity and cost. Looking for a capture balanced prototype, economic and functional, for multigigabit ethernet networks, a hardware system based on programmable devices (FPGAs), particularly in the NetFPGA project, has been implemented. It total costs less than 3100€.

With that aim the development of an own kernel module, data exchange via the PCI-Express system using the Northwest Logic core and the creation of a storage device that ensures multigigabit rate, a level 0 RAID integrated by 8 state solid disks, is required. The bottlenecks of the system are supressed (except for RAID management) in the same way that the processing penalty associated to the TCP/IP stack (due to the FPGA approximation). The memory access is palliated using the huge pages pseudo file system, pages of memory of an unconventional size (greater than 4KB). Using huge pages makes that the transferred size will be greater in each DMA operation.

As a differentiator beyond the purely monetary issue, interception of data at the physical layer in the protocol XGMII is possible. Uncertainties associated with the timing disappear when the application itself is responsible for managing the protocol. The exact time of arrival of each frame is known. Linked to the decision of monitoring the data on the physical level, the hardware design can receive packets with CRC errors in the ethernet layer that would otherwise be discarded.

The document concludes with the expected theoretical results versus actual results, analyzing the differences between both of them, mainly highlighting the performance of RAID where scalability has not been linear. Several points about future work from the 10G functional capturer design are also exposed where 40G capturer is discussed based on the 10G model.

***Index Terms* —** 10g, 10g capture, Dump to disk, FPGA, NetFPGA, PCIe, Huge pages, RAID SSD.

Mi más sincero reconocimiento a todos los miembros del grupo de investigación HPCN ya que sin su ayuda y medios este proyecto no hubiera sido posible.

A su vez, gracias a todos esos amigos y familiares con los que siempre se ha podido contar.

Índice general

Índice de tablas

Índice de figuras

Índice de términos

AG En el sistema de ficheros XFS, grupo individual de gestión. 60

AXI4-Lite Protocolo de comunicación de altas prestaciones, orientado a la transmisión de unos pocos octetos de datos en cada ráfaga. 47

AXI4-Stream Protocolo de comunicación de altas prestaciones, orientado a la transmisión de un flujo continuo de datos. 30, 31, 38, 41, 42, 51, 53, 57

BAR Región de memoria configurable en la FPGA por el sistema, del inglés base address register. 11, 13, 27, 32, 47, 58

BPF Acrónimo para Berkeley Packet Filter. 7

C2S Abreviación para transferencias desde la tarjeta FPGA al sistema anfitrión, del inglés Card to System. 27, 57

CPU Unidad central de procesamiento, del inglés central processing unit. 3–9, 19, 26, 31, 35, 50, 60, 62

DLLP Acrónimo para Data Link Layer Packets o paquete de la capa de enlace en el bus PCI. 54

DMA El acceso directo a memoria permite acceder a la memoria del sistema para leer o escribir independientemente de la unidad central de procesamiento principal. 6, 8, 9, 11, 13, 15, 18–21, 26–28, 30–33, 41, 42, 50, 51, 53, 54, 57, 65

DRAM Memoria de acceso aleatorio dual, es capaz de transferir dos palabras por cada ciclo de reloj. 30

FIFO Propiedad de ciertas estructuras de datos, consistente en que el primer dato insertado coincide con el primero en salir. 12–14, 26, 38, 42, 45, 48

FPGA Dispositivo hardware programable, del inglés Field Programmable Gate Array. 2–5, 7–9, 11, 13, 15, 17, 18, 24, 31, 34, 44–48, 50–52, 57, 58, 65

Gbps Unidad de transferencia, gigabits (10^9 bits) transferidos en un segundo. 1–7, 9, 10, 12, 14, 15, 17–19, 22, 34, 35, 39, 45, 48, 50–54, 57–60, 62–67

GPU Unidad de procesamiento gráfico. 3, 5, 7

IFG Tiempo de inactividad entre el envío/recepción de dos paquetes consecutivos. 22, 45

IO Abreviación para entrada/salida. 4–7, 60, 61, 70

IOCTL Es una llamada de sistema en Unix que permite a una aplicación controlar o comunicarse con el driver de un dispositivo. 26, 28, 30, 49, 58

IP core Bloque lógico de datos. 15, 30

JFS Sistema de archivos de 64 bits con respaldo de transacciones creado por IBM. 60–62

MAC En redes de telecomunicaciones se trata de una subcapa del nivel de enlace en el modelo OSI. 34, 48

MDIO Abreviación del inglés Management Data Input/Output. Es un bus definido en la especificación Ethernet 802.3 para el control de la interfaz independiente del medio, MII.. 36

MSI Método de generación de señales a través del uso de mensajes. 13, 31

MSIX Extensión del protocolo definido por MSI. 13, 31

NUMA Acceso a memoria no uniforme, del inglés Non-Uniform Memory Access. 5, 6

OSI Modelo de red descriptivo, creado por la Organización Internacional para la Estandarización (ISO) en el año 1980. 34, 48, 50, 65

PCAP Formato de almacenado utilizado por la librería libpcap, contracción de Packet Capture. 7, 21, 22, 45, 47

PCI Bus para la interconexión de periféricos, del inglés Peripheral Component Interconnect. 4

PCIe Bus para la interconexión de periféricos ofreciendo altas prestaciones. 4, 8–10, 12–14, 17, 20, 30, 31, 34, 41, 42, 44, 45, 50, 53–57, 59, 63, 64, 67

PHY Nivel 1, o nivel físico, en el modelo OSI de redes de telecomunicaciones. 34, 48

PMA Subcapa dependiente del medio en el nivel físico del modelo OSI. Se sitúa entre la subcapa PMD (physical medium dependent) y la subcapa PCS (physical coding sublayer). 34

QDR Técnica de comunicación de señales, que consiste en transmitir cuatro palabras por cada ciclo de reloj. 19, 51

QPI El Intel QuickPath Interconnect es una conexión punto a punto con el procesador desarrollado por Intel para competir con HyperTransport. 5

RAID Sistema de almacenamiento de datos que usa múltiples unidades de almacenamiento de datos (discos duros o SSDs) entre los que se distribuyen o replican los datos. 17, 20, 21, 55, 56, 58–62, 64, 68, 69

RAM Memoria de acceso aleatorio, del inglés Random Access Memory. 5, 14, 52

S2C Abreviación para transferencias desde el sistema anfitrión a la tarjeta FPGA, del inglés System to Card. 27

SAN Red dedicada al almacenamiento, del inglés Storage Area Network. 3

SFP+ Transceiver insertable en caliente que se emplea para servir de interfaz entre un equipo de comunicaciones y un enlace de fibra óptica. 11

SSD Dispositivo de almacenamiento de datos que usa una memoria no volátil para almacenar datos, en lugar de los platos giratorios magnéticos. 17, 55, 56, 64, 68

TCP Protocolo de comunicación orientado a conexión fiable del nivel de transporte. 4, 5, 8, 35

TLB Es una memoria caché administrada por la MMU (memory management unit), que contiene partes de la tabla de paginación, es decir, relaciones entre direcciones lógicas y físicas. 8

TLP Acrónimo para Transaction Layer Packet o paquete de la capa de transacción en el bus PCI. 10, 54, 55

UIO Módulos a nivel de usuario, del inglés Userspace I/O drivers. 7, 30

WB Política de caché consistente en no volcar los datos hasta que es absolutamente necesario. 61

WT Política de caché consistente en volcar a la unidad de almacenamiento en el mismo momento que se modifican en caché los datos. 61, 62

XAUI Estándar para extender el protocolo XGMII entre las capas MAC y PHY en 10 Gigabit Ethernet. 12, 37, 42, 51, 63, 78

XFS Sistema de archivos de 64 bits con journaling de alto rendimiento creado por SGI. 17, 60–62, 64

XGMII Estándar definido en el IEEE 802.3 para la conexión full duplex de puertos 10 Gigabit Ethernet entre sí y a otros dispositivos electrónicos. 12, 23, 34, 35, 37, 38, 42, 45, 47, 63, 65

1 Introducción

El concepto de red multigigabit ethernet es amplio y hace referencia a la tecnología necesaria para la transmisión de paquetes en la red a tasas superiores a 10^9 bits por segundo o, equivalentemente, 1 gigabit por segundo (Gbps). Su primera formalización fue en el año 2002, en el estándar IEEE 802.3ae-2002 [Soc02], momento en el que se añade la definición para 10 Gbps sobre fibra. La especificación evoluciona a lo largo de los años tanto para anexar nuevos tipos de conexiones, cable axial (twinax) o el par trenzado (UTP), como los protocolos e interfaces para tasas superiores. En el año 2010, se incorpora la descripción de 40 Gbps y 100 Gbps (la aproximación pasa por la utilización conjunta de 10 líneas de 10 Gbps, o 4 de 25, para el segundo caso).

La disminución de los precios de las tarjetas con puertos capaces de gestionar una conexión a 10 Gbps, supone un avance en la expansión y distribución de equipos con esta tecnología. A su vez, el soporte para estos nuevos dispositivos por parte de los sistemas operativos modernos ofrece la posibilidad de su adopción en entornos anteriormente inaccesibles.

La necesidad de monitorizar la información se hace patente. La observación de los datos brinda la oportunidad de detectar comportamientos anómalos en el sistema de enrutado, usuarios malintencionados haciendo un uso inadecuado de los recursos, la recopilación de estadísticas sobre uso de la red que pongan en evidencia problemas asociados al sistema (sobredimensionamiento, cuello de botella no aceptable en hora punta, etc.), monitorización y medida de las prestaciones y disponibilidad de programas software ("application performance monitoring") entre otras aplicaciones.

Las herramientas que permiten la monitorización a una tasa efectiva de 10 Gbps existen. Sin embargo, el principal inconveniente es el desembolso económico que supone adquirirlas. Generalmente, se opta por realizar otros tipo de análisis como los basados en flujos con el correspondiente filtrado del mensaje. En la mayoría de las ocasiones esta simplificación no es un mayor inconveniente ya que aísla, o simplifica, el problema de procesar una gran cantidad de datos.

Un capturador de bajo nivel que permita observar los datos que se transmiten en la red multigigabit sin mediación del sistema es de gran utilidad para ocasiones donde los flujos no aportan información suficiente sobre el caso. Un ejemplo convencional es la actividad de detectar y diagnosticar problemas de rendimiento para mantener el nivel de servicio previsto por otras herramientas de red.

Así pues tras un pequeño estudio de las alternativas actuales se identificarán los problemas asociados a la monitorización de tráfico altas velocidades de transmisión. Se diseñará un prototipo influenciado por las incidencias localizadas por terceros para evitar de esta manera caer en las mismas limitaciones. Se fija el objetivo de crear un capturador de tráfico en redes multigigabit, de 10 Gbps pero fácilmente escalable a otras tasas, económico y accesible.

1.1 Alcance

El principal objeto es la comparativa con otras herramientas del mercado, diseño, desarrollo e implementación de un capturador de tráfico real a 10 Gbps. Para la verificación de la correcta funcionalidad del mismo se usará una herramienta previa diseñada por el alumno (ver sección 4.4.4). Es una aplicación enfocada, principalmente, a un ámbito académico e investigador, aunque cualquier empresa centrada en el desarrollo a altas tasas puede beneficiarse de sus funcionalidades.

El uso de una unidad de coprocesamiento, como un dispositivo hardware programable (FPGA), ofrecerá las interfaces necesarias para la captura de tráfico desde la red, mientras que el sistema principal será el encargado de guardar la información en la unidad de almacenamiento.

No se debe en ningún caso subestimar la complejidad del proyecto, que involucra tanto un desarrollo hardware como el diseño de un driver. Existe un desconocimiento inicial sobre las tecnologías que repercutirán negativamente sobre el tiempo de desarrollo. Por ello, se establecen una serie de hitos intermedios antes de alcanzar la versión final.

1.2 Estructura del documento

En la sección 2 se realiza una comparativa con las herramientas vigentes para el procesado de datos. No todas se dedican en exclusiva a la captura de datos, pero si que deben realizar una actividad de recepción a tasas multigigabit, por lo que muchas de las incidencias detectadas serán extrapolables.

A lo largo del punto 3 se concretan los requisitos que el programa, tanto a nivel software como hardware, debe cumplir. Se detallan tanto los requisitos funcionales, características de la operatividad del sistema, como no funcionales, ligados a aspectos asociados al uso del sistema más que a los comportamientos específicos. Los hitos intermedios prefijados también aparecen bajo este epígrafe.

En el apartado 4 se realiza una presentación descendente sobre el capturador. Se comienza describiendo la máquina física, la implementación del programa cuya funcionalidad es el volcado a disco, para continuar con cuestiones relativas al driver y el diseño hardware. En los apéndices A y B se complementa la información con detalles sobre el sistema operativo y los protocolos de comunicación usados a nivel hardware. Una vez presentada la idea global del capturador se informa sobre otros programas a nivel de usuario cuyo funcionamiento está íntimamente ligado al desarrollo hardware, como la inicialización de las máquinas de estado o la configuración del chip de medio físico.

Conocido el desarrollo del prototipo y los objetivos, falta por evaluar los resultados empíricos. Sin embargo, no se puede hablar de un comportamiento anómalo si se desconocen los datos esperables de manera objetiva. En la sección 5 se realizan ciertas estimaciones previas a nivel teórico para, finalmente, ser comparadas con los datos observables en el diseño. El orden lógico en la realización del proyecto no fue éste. En primera instancia, antes de contemplar cualquier posible adquisición de una máquina fue indispensable obtener una estimación de las prestaciones capaz de ofertar. Sin embargo, a nivel de redacción permite una comparación más directa al agruparlo bajo el mismo punto.

Se finaliza el documento informando de las conclusiones finales y el trabajo futuro a realizar. A pesar de haber completado con éxito el prototipo, sus recursos son explotables para ampliar la herramienta a mayores tasas o crear diseños derivados, de amplia utilidad, como un generador de tráfico sintético a 40 Gbps. Todas estas cuestiones se presentan en el punto 6.

2 Estado de la técnica

2.1 Propuestas para el procesado de datos en la actualidad.

La captura de tráfico en redes multigigabit no es una práctica revolucionaria. Los primeros programas de análisis se remontan hacia el 2003 donde localizamos el proyecto europeo Scampi [Tec03] para la monitorización de redes a 1 Gbps. La evolución sigue el curso natural, adaptando las plataformas existentes hacia las nuevas tasas de transferencia que el mercado exige. Las redes dedicadas al almacenamiento (SANs) y las aplicaciones que hacen uso de "clusters" son pioneras en requerir estas elevadas velocidades. Con su estandarización comienza el diseño de prototipos capaces de trabajar a tasas de 2,5 Gbps [Tec05] y, posteriormente, 10 Gbps [Uni]. Actualmente se investiga por la tasa de los 40 Gbps pero se han dejado entrever previsiones para su expansión en un futuro a 100 Gbps e, incluso, los 400 Gbps [Cor14].

Las limitaciones se hacen patentes. Una gran cantidad de paquetes se necesitan procesar y realizar dicho análisis basándose en una unidad central de procesamiento (CPU) tradicional es una tarea inviable en gran parte de las ocasiones. El uso de unidades de coprocesamiento como unidades de procesamiento gráfico (GPUs) o FPGAs se vuelve, por tanto, corriente en la realización de estas actividades. A medida que se suceden los avances en tecnologías y arquitectura, puede darse el caso en el que cualquiera de las tres alternativas (CPUs, GPUs o FPGAs) resulte adecuada para tratar el problema original.

En este marco donde la CPU de manera aislada sufre carencias para resolver el problema, surgen iniciativas como NetFPGA [Uni], desarrollada en Stanford. Es un proyecto libre, tanto a nivel software como hardware con fines académicos, que permite al usuario elaborar diseños para el procesamiento de paquetes a la velocidad de 10 Gbps. Esta opción será desarrollada en el presente documento a lo largo de la sección 4.1.2. Conviene destacar el hecho de poder construir utilidades de red independientemente del equipo en el que la placa esté conectada.

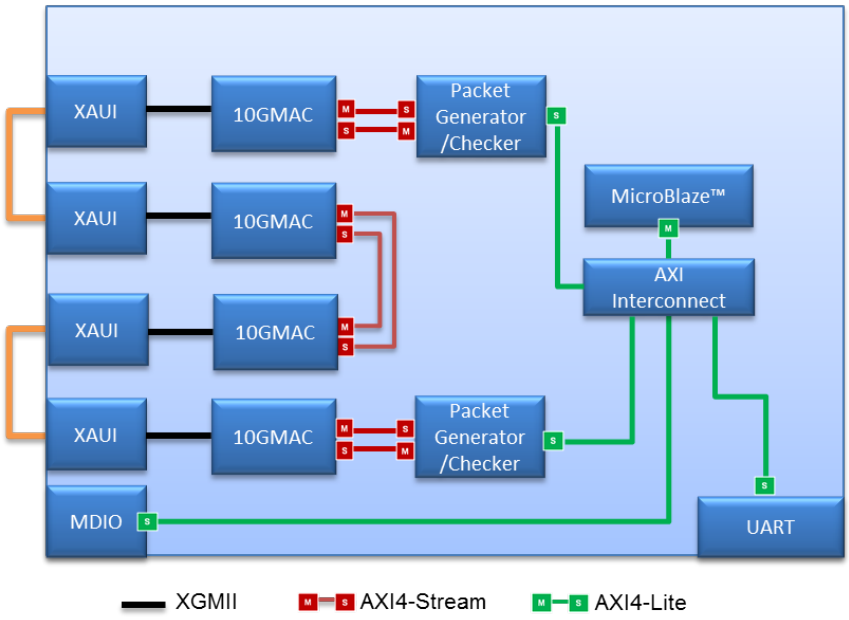


Figura 2.1: Esquema del diseño de referencia de NetFPGA [Uni].

La propia tarjeta de expansión cuenta con cuatro interfaces de red capaces de trabajar a la tasa de 10 Gbps. En el diseño expuesto (ver figura 2.1), se utilizan las cuatro interfaces del

dispositivo de modo que todas queden conectadas. En un extremo se cuenta con un generador de paquetes; en el otro, con una sencilla herramienta de validación. En el caso del diseño de referencia se generan secuencias de números naturales consecutivos que el propio diseño se encargará de comprobar que coinciden con la secuencia lógica al final de la cadena. Las posibilidades de actuación al restringir el diseño al potencial de la FPGA queda limitado. En el caso de un capturador no hay posibilidad de volcar los datos a disco por lo que el diseño se debe extender involucrando en cierta manera a la CPU y al equipo anfitrión. No obstante, hay otra serie de utilidades, imagínese una herramienta que se encargue de filtrar el tráfico basado en reglas cuyo desarrollo puede ser factible de realizar íntegramente a nivel hardware.

Centrándose en el marco de la captura a 10 Gbps, la apuesta de Intel tampoco se hace de esperar, empezando a comercializar tarjetas ethernet enfocadas en la evolución del bus para la interconexión de periféricos (PCI), el bus PCI express (PCIe). Este suceso es acompañado por la mejora de su plataforma en cuanto al manejo de los datos provenientes de la interfaz de red.

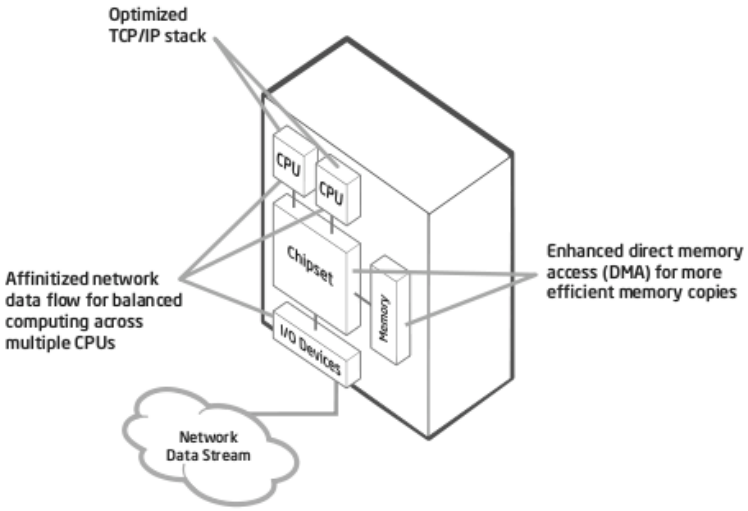


Figura 2.2: Tecnología de aceleración en operaciones de entrada/salida de Intel. Fuente de la imagen [Int06].

En la figura 2.2 se atiende a la exigencia de buscar optimizaciones sobre la arquitectura para estar en condiciones de aplicar el proceso de monitorización de una manera relativamente eficiente. Las aceleraciones implementadas en operaciones de entrada/salida (IO) son:

- Procesamiento paralelo en el protocolo de control de transmisión (TCP). Menores penalizaciones por parte de sistema y una mejora de la eficiencia en procesamiento de la pila TCP gracias a la posibilidad de las CPUs actuales de ejecutar varias instrucciones por ciclo. Se pretende cargar previamente al paquete la cabecera y realizar el movimiento de datos de forma paralela.

- Explotar la afinidad de las CPUs para el tratamiento de los flujos de datos, necesidad que también se detectará de manera concurrente con la aplicación "PacketShader" [SH11].

- Copias asíncronas a mayor velocidad.

- Mejorar el procesamiento de la pila TCP/IP. En este caso se opta por implementar caminos diferentes para el control y los datos, favoreciendo el análisis de los paquetes desde la cabecera hasta el "payload".

Estas mejoras están claramente orientadas a disminuir una carga de trabajo, nada despreciable, según las estimaciones [Int06] y reflejadas gráficamente en la figura 2.3.

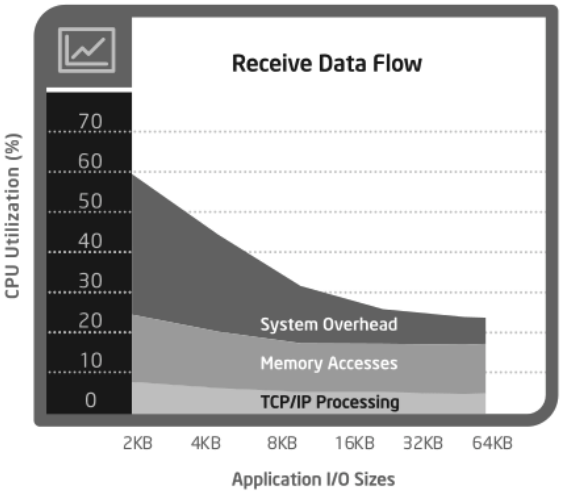


Figura 2.3: Utilización de la CPU en operaciones de IO. Fuente de la imagen [Int06].

El driver Intel DPDK [Int] puede utilizarse conjuntamente en una máquina con la arquitectura previamente mencionada para la captura de tráfico [RL13]. El trabajo realizado bajo este entorno parece satisfactorio a redes de 10 Gbps donde la monitorización se realiza sin pérdida de información, es decir, de paquetes. En cualquier caso, las estaciones utilizadas para tal actuación son equipos con gran potencia. En el caso de [RL13] para construir el capturador de tráfico se hace uso de una máquina que integra los recursos hardware mostrados en la tabla 2.1.

CPU	**Memoria de acceso aleatorio (RAM)**	**Tarjeta de red**
Intel Xeon E5-2609 con 10Mb de caché de nivel 3 a 2.40 GHz	12 Gb	Intel Corporation 82599EB 10-gigabit

Tabla 2.1: Recursos hardware que aseguran la captura de tráfico a 10 Gbps sin el uso de unidades de coprocesamiento.

Paralelamente al desarrollo mediante FPGAs, se encuentran las GPUs. En la figura 2.4 se presenta un esquema de la arquitectura utilizada en "PacketShader" [SH11]. La aplicación implementa una herramienta de enrutado capaz de trabajar en redes multigigabit. En el trabajo citado, la tasa de 10 Gbps es alcanzada holgadamente. La arquitectura hace uso de dos procesadores independientes conectados a través de quick path interconnect (QPI), un total de 8 interfaces de red a 10Gbps y dos GPUs. El diseño se cataloga bajo el tipo de acceso a memoria no uniforme (NUMA).

El flujo de datos proveniente de las interfaces es procesado paralelamente por aquella GPU que comparte el mismo hub IO. Las aplicaciones se optimizan para considerar posibles penalizaciones temporales que podría tener el uso de la memoria que está conectada directamente a otra CPU o, los propios periféricos 'lejanos'. Es importante notar que los desarrolladores han debido realizar ciertas modificaciones a la configuración por defecto del software y que, en cierto modo, podrán adaptarse a la aplicación propuesta en este trabajo:

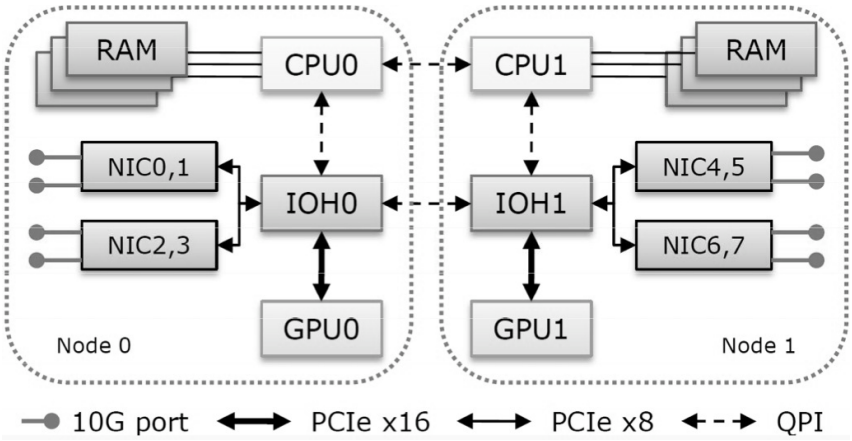


Figura 2.4: Esquema de utilidad de enrutado basada en GPUs. Fuente de la imagen [SH11].

- Mejoras en la pila de red del sistema Linux. La pila TCP en el kernel del sistema operativo mantiene buffers para cada paquete individualmente. Este hecho crea dos inconvenientes principalmente:

  1. La reserva y liberación de memoria provoca una carga extra en los módulos del núcleo encargados del manejo de la memoria. En el caso de redes a 10 Gbps aproximadamente equivale a decenas de millones de reservas por segundo.
  2. Cada buffer por paquete lleva asociado información útil para las distintas capas (metadatos), que en la versión 2.6.28 presentaban una longitud total de 208 bytes.

- Implementación de un nuevo esquema de reserva de memoria. Ya se ha explicado el sistema que se aplica a cada paquete individualmente. En [SH11] se propone generar dos buffers de gran tamaño inicialmente. En el primero se alberga información relativa a los metadatos (entre los cuales se eliminan campos raramente usados); en el segundo, se implementa una cola circular de tal manera que no sea necesario reservar/liberar memoria para cada paquete recibido. La reserva inicial de estas regiones colabora con el hecho de que las operaciones de acceso directo a memoria (DMA) sean eficientes (comúnmente se tratará de grandes regiones de memoria contiguas).

- Procesamiento por lotes en diferentes niveles:

  1. A nivel hardware. La interfaz de red agrupa múltiples paquetes en una única transacción para una utilización más provechosa del ancho de banda de IO.
  2. A nivel de driver y a nivel de usuario. El procesamiento por bloques logra disminuir el número de invocaciones de funciones y las sincronizaciones entre operaciones.

- Escalabilidad en sistemas multicore. Que el sistema sea capaz de adaptarse y mejorar su rendimiento de manera aceptable ante la incorporación de nuevos cores impone dos objetivos principales:

  1. Balanceo equilibrado de la carga.
  2. Escalabilidad lineal con el aumento de cores.

- Escalabilidad de la arquitectura NUMA. En un sistema NUMA el acceso a memoria depende de la localización física del módulo. Existe conexión directa a la memoria 'local' mientras que el acceso a la memoria remota requiere de al menos un salto extra a través

de la CPU más cercana a ese módulo. Las operaciones de DMA de un dispositivo a un nodo remoto involucra múltiples hubs IO y reduce el rendimiento.

Los ejemplos ya mencionados proporcionan un punto de partida para diferenciar el abanico de opciones que se despliega a la hora de seleccionar una posible arquitectura para solventar el problema propuesto. Sin embargo, ninguna de las tres utilidades, ni la opción que se apoya en GPUs/FPGAs, ni la versión que hace uso exclusivo de la CPU, conservan el contenido en disco.

Esto impone una limitación nada despreciable puesto que no hay una estrategia sencilla y válida en todas las ocasiones para realizar el almacenamiento sin que ello perjudique a la tasa de procesamiento. Se presta atención a la utilidad "n2disk" [nto14]. La arquitectura para la que se ha pensado esta herramienta es la reflejada en la figura 2.2. Es decir, requiere de una interfaz de red de Intel con soporte para tasas de 10 Gbps aunque la herramienta en las pruebas realizadas no llega a aprovechar la totalidad del ancho de banda (ver tabla 2.2).

Tamaño del paquete	**Rendimiento**
Tamaño fijo: 64B	7,94 Mpps, 5,33Gbps
Tamaño fijo: 128B	4,64 Mpps, 5,5Gbps
Tamaño fijo: 512B	1,29 Mpps, 5,5Gbps
Tamaño variable: 64-1500B	857 Kpps, 5,5Gbps

Tabla 2.2: Rendimiento del programa "n2disk" [nto14].

La novedad de esta herramienta, más que las tasas ofrecidas, son las utilidades que favorecen su adopción al utilizar los estándares en formatos de capturas:

- Posibilidad de trabajar con ficheros en formato Packet Capture (PCAP).
- Posibilidad de aplicar filtros Berkeley Packet Filter (BPF), tanto en la captura como en las posibles consultas posteriores a los datos.
- Utilización de múltiples hilos para la explotación de sistemas multicore. Al menos un hilo se dedica a captura y otro a escritura en disco: modelo productor-consumidor.
- Política de escritura directa en disco ("Direct-IO"). Modifica el comportamiento por defecto del sistema cuando se abre un fichero en un dispositivo externo para su escritura. Típicamente se genera un buffer en memoria principal, donde se albergan los datos para su posterior escritura evitando un número mayor de operaciones sobre el medio externo. Para aplicaciones convencionales esta aproximación puede ser plausible. Sin embargo, en programas donde la principal finalidad es obtener un alto rendimiento, la penalización de esta doble escritura (en memoria principal y en la del medio externo) podría ser mayor que la demora producida en el caso en el que el programa software realizase las operaciones de escritura directamente contra el dispositivo. En este programa los desarrolladores han optado por la supresión de las posibles copias intermedias para asegurar una mayor tasa de escritura dado que se esperan transferir grandes porciones de datos y el efecto del buffer intermedio no tiene una justificación clara.
- Utiliza módulos a nivel de usuario (UIO), es decir, drivers capaces de acceder a las tarjetas de red para la captura/transmisión de paquetes de/desde el espacio de usuario sin intervención del kernel. En la figura 2.5 se muestra el diseño del driver a nivel de usuario.
- Generación de índices que posibiliten la consulta de paquetes dentro del fichero generado sin necesidad de realizar una búsqueda secuencial (consultar la descripción del formato PCAP).

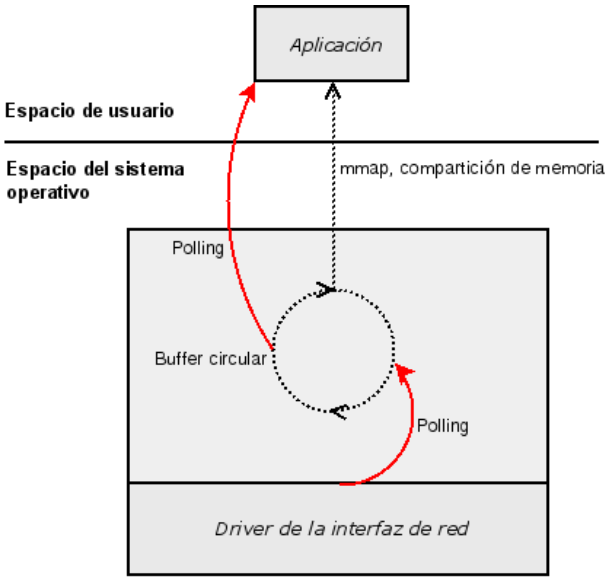


Figura 2.5: Esquema de la comunicación con la interfaz de red en "n2disk".

- Uso de páginas de un tamaño superior al habitual. El kernel de linux típicamente emplea páginas de memoria de tamaño 4KB. Sin embargo, provee de una interfaz para la reserva de páginas de un tamaño mayor comúnmente llamadas "páginas enormes" de su traducción del inglés "huge pages". El uso de huge pages ofrece dos ventajas principalmente:

  1. Grandes regiones de memoria contigua pueden ser reservadas en el arranque del sistema que de otro modo fallarían.
  2. Reduce la carga del sistema. En el sentido de que la memoria Translation Lookaside Buffer (TLB) contiene traducciones de direcciones virtuales a físicas. Utilizar una gran cantidad de memoria con el tamaño por defecto conlleva un procesamiento extra para manejar las entradas del TLB.

2.2 Conclusión

Las alternativas son variadas en cuanto a la temática de análisis y procesamiento de datos en redes multigigabit se refiere. A grandes rasgos se distinguen dos tendencias: herramientas que se apoyan en el uso de unidades de coprocesamiento y aquellas otras que, por el contrario, dependen de la eficiencia y capacidad de la CPU del sistema.

En este trabajo desarrolla un capturador de tráfico con el equipo más sencillo posible. Es decir, se prima la optimización de los recursos disponibles con el fin de obtener el mayor rendimiento posible. Se pretende diseñar una utilidad de monitorización a partir del proyecto NetFPGA [Uni]. Utilizar FPGAs para el desarrollo suprimirá, por ejemplo, los costes derivados de la pila TCP/IP. Permitirá, a priori, utilizar una máquina con recursos más modestos para la ejecución de las pruebas y, la evolución hacia tasas superiores no parece reñida con las especificaciones del proyecto (el hardware de NetFPGA utiliza exclusivamente PCIe de primera generación). El uso de páginas de memoria no estándar, en conjunto con el uso de DMA para la transmisión del contenido hacia las mismas permitirá un acceso más óptimo a memoria.

Por último, conviene mencionar la inexistencia de ninguna herramienta que capture a nivel de la capa física, a nivel XGMII, todas lo hacen a nivel superior por lo que aparecen incertidumbres en la temporización y se descartan paquetes con CRC incorrecto en el nivel Ethernet.

3 Especificación de requisitos

3.1 Definición del proyecto

3.1.1 Objetivos y funcionalidad

En la presente sección se describen las características generales con las que el capturador de tráfico debe cumplir.

Perspectiva del producto

La complejidad del desarrollo es palpable por varias razones fundamentales:

- Se requiere de una única versión producto, que tras unas mínimas configuraciones pueda ser adaptada a distintos equipos.
- Dada la necesidad de realizar un procesamiento a tasas multigigabit la eficiencia es una de las prioridades. En el caso de no cumplir con la tasa de 10 Gbps no se puede aceptar el trabajo como plenamente válido.
- Debe ser capaz de funcionar de manera independiente a la CPU, de tal modo que la mayor carga de trabajo resida en el diseño hardware, liberando de trabajo a la CPU gracias a la utilización de operaciones DMA.
- Tiene que cumplir con las restricciones de presupuesto. En la medida de lo posible se persigue alcanzar un proyecto de costo reducido. Los problemas aparecidos deben ser solventados mediante el acercamiento al problema por otros caminos más que por la mejora del equipo. Nótese, en cualquier caso, la necesidad de poseer una máquina en la que sea factible realizar la captura a tal velocidad.

A estas tareas hay que añadir la penalización de que se trata de un proyecto que unifica tanto un diseño a nivel hardware como a nivel de software. Típicamente el desarrollo hardware es más lento. Realizar un proyecto que involucra ambas áreas (desarrollo hardware y software) acentúa la necesidad de realizar unas pruebas de integración más exhaustivas. A nivel software la necesidad de un driver propio, probablemente desemboque en una demora de la realización.

La estructura simplificada del proyecto se muestra en la figura 3.1. Se dispone de un equipo con una tarjeta NetFGPA [Uni], con soporte para la tecnología PCIe de primera generación. En una de las interfaces multigigabit se monitorizarán los datos que serán volcados a un sistema de almacenamiento por el ordenador anfitrión una vez que hayan sido puestos a su disposición por la FPGA.

Figura 3.1: Arquitectura del proyecto.

Funciones del sistema

Las funciones que se deben desarrollar son:

(a) Capacidad de interpretar la información proveniente de la red a nivel físico en la plataforma hardware. De aquí en adelante este subsistema se identificará como recepción.

(b) Transmisión de los datos al ordenador anfitrión. Éste es el subsistema de comunicación.

(c) Módulo encargado de recibir los datos provenientes del subsistema de comunicación y ponerlos a disposición del programa de usuario. Subsistema de driver.

(d) Programa a nivel de usuario cuya funcionalidad sea el guardado de los datos en un medio físico. Subsistema de almacenamiento.

(e) Mecanismo de detección de errores y degradación de rendimiento. Subsistema de reporte.

(f) Generación de estadísticas. Subsistema de información.

(g) Configuración del diseño hardware a través de programas de usuario. Subsistema de configuración.

Subsistema de recepción: el propósito fundamental de la aplicación es permitir un manejo eficiente de la interfaz de entrada. Por manejo eficiente, debe entenderse que en todo momento el sistema debe ser capaz de auditar el flujo de datos proveniente de la red sin pérdida de datos. No se entiende por pérdida de datos el descarte de información debido a la monitorización en redes que trabajen a una tasa superior a la que la interfaz de red del equipo soporte. Debe permitir, por tanto, realizar las siguientes tareas:

- Detectar y almacenar el flujo de datos en redes multigigabit (10 Gbps) capturando tramas mal formadas o con errores mientras se asegura una monitorización precisa del tiempo de llegada de paquetes.

Subsistema de comunicación: dados una serie de bytes, el sistema debe ser capaz de encapsularlos para su envío mediante PCIe. Es decir, debe crear paquetes en formato Transaction Layer Packet (TLP) y formar trazas conforme a los distintos formatos que intervienen en las comunicaciones mediante PCIe (debe interpretar tanto los formatos para lecturas/escrituras en memoria como el formato para asegurar la recepción de un paquete). En la figura 3.2 se muestra el aspecto básico de un paquete TLP de escritura en memoria del anfitrión. El subsistema debe, por tanto,

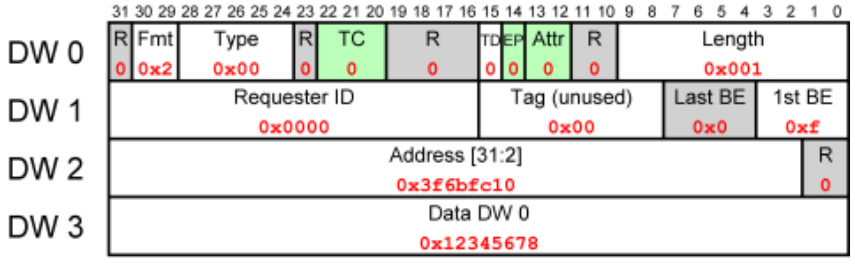


Figura 3.2: Formato paquete TLP básico. Fuente [Xilb].

- Ceñirse al estándar de PCIe 1.0 [PS05].

- Permitir la comunicación de los datos mediante operaciones DMA.
- Notificar mediante el uso de interrupciones al subsistema de driver la presencia de nuevos datos que tratar.

Subsistema de driver: el proyecto anexiona un módulo para el sistema operativo, con conocimiento de la existencia del subsistema de comunicación. Interacciona con el sistema anterior para transmitir la información al posible programa de usuario. A su vez, es el punto de conexión con el sistema hardware por lo que la lectura/escritura en los registros de configuración será mediada por el driver. Por tanto, las funcionalidades que ofrece son:

- Configuración de la capa de nivel físico de la FPGA, dado que el estándar no es único y es necesario realizar una configuración específica según el tipo de dispositivo que sirva de interfaz con la fibra (aquellos basados en enhanced small form-factor pluggable (SFP+) deben ser soportados).
- Configurar las transferencias mediante DMA en el motor hardware. Debe además determinar las direcciones de memoria donde la lectura/escritura será realizada.
- Atención a las interrupciones generadas por el subsistema de comunicación.
- Interactuar con el diseño de usuario.

Subsistema de almacenamiento: una vez que se tengan los datos en el programa a nivel usuario tras la previa puesta a disposición por parte del driver, es necesario que los datos se almacenen para su posterior consulta. El formato del fichero guardado debe estar pensado de tal modo que tanto los datos como los respectivos momentos de llegada puedan ser obtenidos fácilmente. Se espera que:

- El sistema de almacenamiento asegure la tasa global del sistema.
- El formato de los ficheros sea lo más estándar posible o, en su defecto, fácilmente adaptable.

Subsistema de reporte e información: el driver basándose en el soporte por parte del diseño hardware ofrece la posibilidad de consultar los estadísticos sobre el uso de los motores DMA. A su vez, es posible consultar el estado actual distinguiendo varios posibles casos: parado, en captura y por debajo del rendimiento óptimo. Se ofrece la posibilidad de:

- Consultar el estado actual de la máquina de estados del diseño hardware. Posibles errores se verán reflejados por un estado inválido o inesperado.
- Consultar información específica sobre el rendimiento.

Subsistema de configuración del diseño: el driver provee de las herramientas para la lectura/escritura en el base address registers (BAR). El diseño hardware modifica en función de los parámetros especificados su comportamiento habitual. Entre estos parámetros de configuración se encuentran las instrucciones de inicio/fin de la captura y serán desglosadas en los requisitos funcionales.

3.2 Catálogo inicial de requisitos

A continuación se dividen los requisitos que deberá presentar el modelo final. Se distinguen en dos grupos claramente diferenciados: requisitos funcionales y requisitos no funcionales. En el primer grupo se exponen aquellos criterios relacionados con el comportamiento del sistema; en el segundo, características requeridas del sistema (o del servicio prestado) que señalan una restricción del mismo.

3.2.1 Requisitos funcionales

Subsistema de recepción

RF(1) Se proporciona soporte para adaptadores de Ethernet ópticos, SFP+ SR y SFP+ LR de 10 Gbps, además de los adaptadores SFP+ direct attach copper cable, SFP+ Cu de 10 Gbps.

RF(2) El diseño implementa, al menos, los siguientes estados:

- Parado. Si no se está operando.
- Capturando. Se está monitorizando la interfaz de red multigigabit.
- Error. Ha ocurrido alguna anomalía en el proceso en cuyo caso se detiene cualquier acción que pudiera estar ejecutándose.

RF(3) Los datos capturados se almacenarán en una cola first in first out (FIFO) implementada con Block RAMs.

RF(4) Debe existir un control sobre los datos capturados. En el caso de no poder atender nuevos flujos (por ejemplo la cola esta llena) la máquina de estados del diseño hardware se dirigirá al estado de error.

RF(5) Se debe soportar el protocolo Attachment Unit Interface (XAUI).

RF(6) La captura se realiza a nivel del protocolo 10 Gigabit Media Independent Interface (XGMII).

RF(7) No se ignorarán paquetes con errores de forma, o código de redundancia, en el nivel de enlace y superiores.

RF(8) Se realiza la monitorización de los tiempos de llegada entre dos paquetes sucesivos.

Subsistema de comunicación

RF(9) Debe controlarse el uso de diferentes frecuencias de reloj sin que ello produzca una alteración en la integridad de los datos. La motivación del uso de distintas señales de reloj se debe a que el protocolo XAUI a 10 Gbps requiere de una onda a 156.25 MHz y la señal de PCIe únicamente 125 MHz.

RF(10) Los datos se transmiten al sistema anfitrión mediante DMA. Debe implementar la técnica scatter-gather, que permita la transmisión de regiones no contiguas de memoria con una única configuración.

RF(11) Periódicamente se transmiten los datos al driver. Este proceso puede ocurrir por:

- Alcanzar una cota sobre el tamaño de datos capturados en la cola del subsistema de recepción.

- Haber monitorizado el número de paquetes prefijado.

El segundo umbral se configura a través de un registro de 128 bits del BAR 1. El primero queda a elección de la implementación del diseño hardware.

RF(12) Se generarán interrupciones message signaled interrupt (MSI)/message signaled interrupt extended (MSIX) para notificar de cada transferencia completada.

Subsistema de driver

RF(13) No modificará los datos obtenidos del bus PCIe.

RF(14) Creará un dispositivo de caracteres que permita a un programa de usuario la interacción con el driver: lectura/escritura en registros y lectura/escritura mediante operaciones de DMA.

RF(15) Creará un fichero virtual bajo el directorio "/proc/" donde poder consultar las estadísticas de trabajo: ocupación de la FIFO, estado actual de la máquina de estados y número de bytes transferidos tanto en su totalidad como en el último segundo.

RF(16) Ofrecerá modos de operación tanto bloqueantes como no bloqueantes para la lectura/escritura de datos desde/hacia la FPGA.

RF(17) Implementa el manejo de "huge pages".

Subsistema de almacenamiento

RF(18) Asegura un modelo de "zero-copy". La información obtenida del diseño hardware no se duplica en memoria principal durante todo el proceso de captura hasta su volcado a disco.

RF(19) El formato del fichero volcado contiene los siguientes campos:

- Tamaño en bytes del paquete.
- Hora de recepción.
- Contenido del paquete.

Subsistema de reporte e información

RF(20) Registra los errores de la plataforma, tanto los ya mencionados a nivel del diseño hardware (reflejados a través de la máquina de estados) como los asociados a errores en operaciones de DMA. La detección consiste en la actualización de los registros propios de la FPGA (registro de 128 bits del BAR 1).

RF(21) Registra el tiempo que el motor DMA ha estado activo durante el último segundo, tiempo que ha estado esperando a nueva información en el último segundo y número de bytes transferidos en el último segundo. Estos tres valores se almacenan en el fichero virtual bajo el directorio "/proc".

RF(22) En un registro de 128 bits del BAR 1 se alberga la ocupación de la cola FIFO expresada en palabras de 64 bits.

Subsistema de configuración

RF(23) El sistema permite configurar el inicio y la parada del módulo hardware.

RF(24) El sistema permite solicitar un reinicio del diseño hardware.

RF(25) El sistema permite configurar la ocupación mínima de la FIFO antes de comenzar a transmitir datos desde el hardware al anfitrión.

RF(26) Se permite seleccionar el nombre del fichero de salida, así como su ubicación.

3.2.2 Requisitos no funcionales

Rendimiento

RNF(1) El sistema soportará la tasa de 10 Gbps, tanto en captura en la interfaz de red como en volcado al sistema de almacenamiento.

RNF(2) El sistema de captura no sobrecargará la unidad central de procesamiento. Es decir, no utilizará más del 50 % de los recursos ofrecidos por ésta.

Usabilidad

RNF(3) La aplicación debe ofrecer la ayuda necesaria, ya sea mediante documentación o asistentes en ejecución, para que otros usuarios distintos de los desarrolladores puedan utilizarla sin restricciones por dificultad.

RNF(4) Exportación a posteriori, es decir, después de haber realizado la captura, a un formato genérico como PCAP.

Escalabilidad

RNF(5) El sistema será escalable. Debe ser fácilmente adaptable a equipos con mayores prestaciones y mejores tecnologías como pudieran ser equipos que integrasen PCIe de tercera generación o cuyo sistema de almacenamiento fuera distinto.

Coste

RNF(6) Una de las premisas del proyecto es construir un capturador de bajo coste. El sistema debe ser capaz de ejecutarse en ordenadores que cuenten con la tecnología PCIe de primera generación (al menos 8 líneas disponibles), 8 Gb de memoria RAM libres y un sistema de almacenamiento que asegure la escritura a la tasa de 10 Gbps.

Estabilidad

RNF(7) Se pretende diseñar un proyecto capaz de estar trabajando durante largos periodos ininterrumpidos de tiempo. Por tanto, es un requisito indispensable que el sistema no se cuelgue ni disminuya su rendimiento a lo largo de estas jornadas operativas.

Compatibilidad

RNF(8) El diseño debe funcionar en máquinas con sistema operativo Linux, con versiones del kernel comprendidas entre la 2.6.32 y la 3.13.

Estilo

RNF(9) Los módulos deben estar escritos de manera clara, legible y documentada, de forma que se pueda asegurar su reutilización posterior por desarrolladores independientes.

3.3 Modelo de ciclo de vida

El ciclo de vida incremental e iterativo es aquel en el que se va liberando parte del producto periódicamente, y cada entrega es una mejora respecto a la anterior; cada fase (requisitos, análisis, diseño, etc.) se realiza varias veces. Lo cual difiere del desarrollo en cascada, donde las fases del ciclo de vida (requisitos, análisis, diseño, etc.) se realizan (en teoría) una única vez, y el inicio de una fase no comienza hasta que termina la fase que le precede.

Aunque los requisitos están inicialmente fijados, la compartición de los prototipos con otros usuarios aporta información sobre nuevas necesidades que son requeridas. El modelo de ciclo de vida incremental permite aplazar la toma de decisiones hasta el momento en el que se adquiere una mayor experiencia con el entorno y será el modelo a seguir.

El capturador de tráfico se construye prácticamente desde la nada, aunque no se puede omitir que se reutilizan componentes ajenos, concretamente en la parte del diseño hardware donde se utilizarán bloques lógicos de datos (IP cores) de terceros. Para la elaboración del capturador se establecen los siguientes incrementos inicialmente que coinciden, generalmente, con la incorporación de un nuevo elemento ajeno y/o su manejo:

- Diseño hardware cuya funcionalidad sea la generación de números naturales consecutivos y su transmisión mediante DMA al sistema anfitrión. Los datos pueden ser recepcionados con el módulo libre del proyecto NetFPGA [Uni]. Esta utilidad se corresponde con el subsistema de comunicación.

- Creación de un driver propio (más capa de middleware y programa de usuario) capaz de recibir datos desde la tarjeta FPGA mediante transacciones DMA y escribir su contenido a disco. Se desarrolla el subsistema de driver, y subsistema de almacenamiento.

- Desarrollo del driver para que implemente la posibilidad tanto de recibir como de transferir datos a la FPGA. Mejora del subsistema driver.

- Elaboración de un diseño hardware capaz de capturar tráfico de las interfaces de red. Creación del subsistema de recepción.

- Mejora del diseño hardware que sea capaz de distinguir los datos de los bytes de control. Refinamiento del subsistema de recepción.

- Refinamiento del diseño hardware de modo que se concatenen campos extra como el tiempo de llegada o el tamaño de cada paquete. Se completa el subsistema de recepción, se anexiona el subsistema de configuración del diseño y el subsistema de reporte e información.

Durante cada una de las etapas es requisito indispensable verificar que el rendimiento obtenido por el sistema es el idóneo para la monitorización a 10 Gbps. No tiene sentido considerar un hito como finalizado si no se cumple esta restricción.

3.4 Resumen

En la presente sección han quedado definidos los requisitos que el capturador de tráfico basado en hardware para redes multigigabit debe poseer. El modelo de vida prefijado es un modelo incremental e iterativo, donde haya posibilidad de revisar los aspectos del producto tras interpretar nuevas necesidades.

A partir de los requisitos iniciales se desarrolla un prototipo, del que se profundizará y explicará en más detalle sus características en las siguientes secciones. Mediciones de prestaciones también son llevadas a cabo, así como las estimaciones óptimas teóricas.

4 Sistema elaborado

En esta sección se explica la arquitectura desplegada, los programas empleados y las configuraciones aplicadas para el desarrollo de un capturador de tráfico en redes multigabit ethernet hasta una tasa de 10 Gbps.

4.1 Equipo físico

Para la realización de las pruebas se cuenta con el siguiente material físico.

- Placa base Supermicro X10SL7-F ($249.99).
- Procesador Intel(R) Xeon(R) CPU E3-1230 v3 @ 3.30GHz ($249.99).
- Cuatro módulos de memoria Kingston DDR 3, 8GB a 1600 MHz (4x$96.99).
- Ocho discos solid state drive (SSD) Samsung SSD 840 EVO (8x$204.99).
- Disco duro Seagate Barracuda 7200 rpm 500GB SATA3 ($59.99).
- NetFPGA 10G (4x10G, aproximadamente $1500 para universidades).

El capturador elaborado ofrece flexibilidad como para ejecutarse en distintos entornos. Se ha primado que la instalación en un nuevo centro no imponga restricciones por lo que no deberían existir problemas de compatibilidad para la instalación en un equipo cuyo sistema operativo sea una versión GNU/Linux comprendida entre la revisión 2.6.32 y 3.13 (última versión del kernel a la fecha de redacción del documento). Si se opta por la compilación de una versión propia del kernel se recomiendo consultar el apéndice A.

Como sistema de referencia se escoge CentOS 6.5 [Cen14]. Por defecto ya se encuentra habilitado el sistema de "huge pages" con lo que el diseño de usuario podrá funcionar sin modificaciones adicionales. Adicionalmente, el sistema de ficheros Extended File System (XFS) está soportado tras la instalación de xfsprogs. Para la compilación del módulo kernel propio es necesario instalar el paquete kernel-devel.

Para la creación de un Redundant Array of Independent Disks (RAID) software, la herramienta mdadm figura como preinstalada. En el apéndice A aparece como crear un RAID0 a nivel software. El dispositivo NetFPGA debe estar conectado en un puerto de expansión que cumpla con el estándar de PCIe de primera generación y que disponga de, al menos, 8 líneas.

4.1.1 Herramientas necesarias para interactuar con NetFPGA

Para la programación de la FPGA es necesario disponer de las herramientas de Xilinx Lab Tools y Digilent (Digilent Runtime y Digilent Utilities). Los drivers de cable de la suite de Xilinx parecen presentar incompatibilidades durante la instalación en sistemas operativos Linux. Por ello se recomienda, de manera alternativa, usar los paquetes de Digilent. Para el correcto funcionamiento de los mismos, es necesario instalar mediante el gestor de paquetes de la distribución las librerías abiertas fxload, libusb y los "headers" de esta última (libusb-dev).

A su vez, es necesario crear un enlace simbólico de la librería libusb al directorio donde la herramienta de impact la busca por defecto: /usr/lib/ (en caso de no localizarse el paquete informará con un error que no es capaz de detectar el dispositivo). La ubicación predeterminada es /usr/lib/x86_64-linux-gnu/libusb-0.1.so.4.

4.1.2 Plataforma NetFPGA

La plataforma NetFPGA es un proyecto abierto que involucra tanto un desarrollo basado en FPGAs como un módulo para el kernel de linux (o para sistema operativo windows) con las funcionalidades básicas para interactuar con el hardware. Tras la evaluación del diseño inicial (figura 2.1), las limitaciones se hicieron patentes y la sencillez del diseño en cuanto al abanico de operaciones permitidas y las prestaciones ofertadas se antojaron insuficientes.

Por ello, se decide plantear sobre esta plataforma hardware un diseño alternativo, pero sobre un elemento físico de amplia distribución, la FPGA de Xilinx, Virtex-5 XC5VTX240T-2FFG1759C. Se desarrolla un driver propio, con opciones para el manejo de huge pages (ver sección 4.1.3) que limita el sistema de pruebas a un entorno linux pero que permite explotar recursos que de otra manera no serían tan accesibles.

Tras observar las limitaciones del core DMA, que asegura unas tasas de transferencia desde la tarjeta hacia el sistema no superiores a los 7 Gbps, se opta por utilizar una alternativa propietaria que asegure obtener las velocidades necesarias para elaborar un capturador a 10 Gbps.

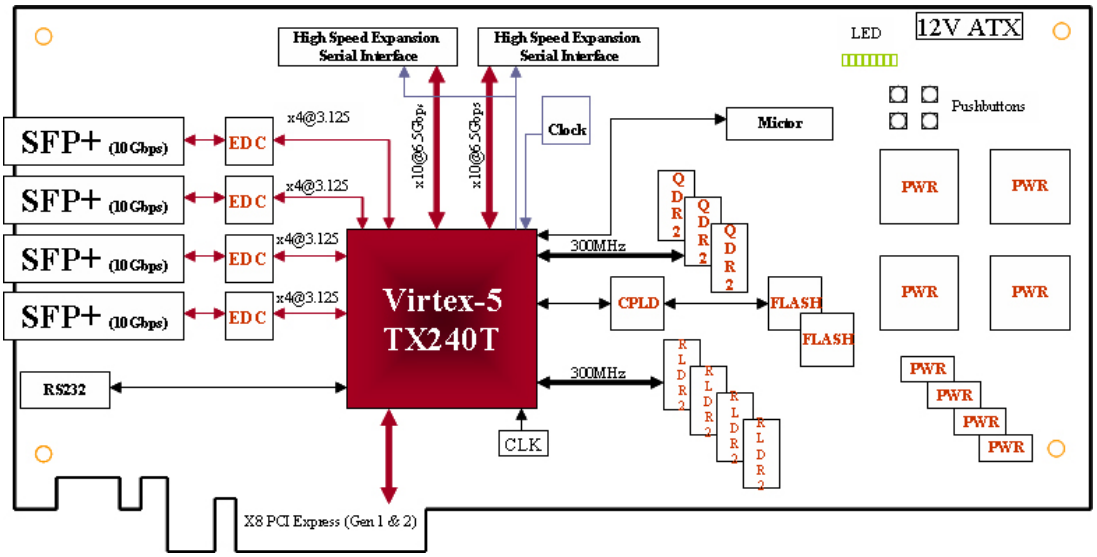


Figura 4.1: Diagrama de bloques de la FPGA utilizada. Fuente [Glo].

Las especificaciones de la tarjeta hardware son las siguientes:

- Cuatro interfaces SFP+ (usa 16 RocketIO GTX transceptores y 4 dispositivos Broadcom EDC(electronic dispersion compensation)).
- Soporte para los modos de 10 Gbps y 1 Gbps.
- PCI Express Gen 1 x8 (250 MB/s hábiles en cada línea).
- Veinte transceptores GTX seriales configurables.
- Tres dispositivos de memoria: x36 Cypress QDR II (CY7C1515JV18).
- Cuatro dispositivos de memoria: x36 Micron RLDRAM II (MT49H16M36HT-25).
- Dos dispositivos (de almacenamiento de alto rendimiento) Xilinx Platform XL Flash de 128MB cada uno.

- Una unidad CPLD XC2C256 de Xilinx diseñada para la gestión de la memoria flash de configuración.
- Un puerto serie DB9 (RS232).

En la figura 4.1 aparece el esquema de la placa utilizada. Para el diseño de captura, se utilizará exclusivamente el modo de 10 Gbps sobre una interfaz SFP+. El uso de más módulos externos se descarta y en la sección 5.1 se realiza un análisis de la viabilidad de utilizar las memorias quad data rate (QDR). Se terminará aplicando una opción basada en la memoria interna del dispositivo, block RAMs.

4.1.3 Configuración del arranque

Para que el sistema pueda realizar operaciones DMA de manera eficiente se hace uso de páginas de un tamaño no estándar. Para poder acceder a su manejo es primordial realizar unos ajustes previos a las opciones de arranque del sistema. Es necesario añadir a la entrada de inicio deseada los siguientes argumentos:

- *default_hugepagesz=1G*. Establece 1GB como el tamaño por defecto para las páginas de tamaño no convencional.
- *hugepagesz=1G*. En ciertos sistemas se soportan páginas de distintos tamaños. Para indicar el tamaño usado se utiliza este parámetro.
- *hugepages=8*. Número de páginas reservadas inicialmente por el sistema operativo y que podrán ser solicitadas durante la ejecución del programa.

Conviene añadir también a las opciones de arranque el aislamiento de una serie de CPUs. Se recomienda aislar dos: una en la que se ejecutará el programa de usuario; otra, para la función manejadora de interrupciones del driver. Esto se puede lograr con la sentencia mostrada en el cuadro 4.1

```
1  isolcpus=0,1
```

Cuadro 4.1: Aislamiento de una lista de CPUs, 0 y 1 en el ejemplo, durante el arranque.

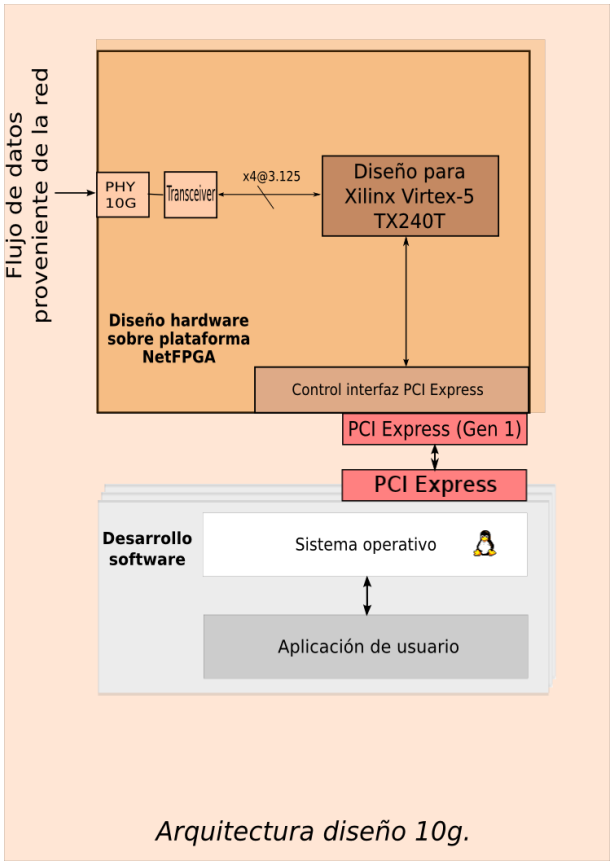
4.1.4 Modificaciones en la configuración de la BIOS.

Las opciones por defecto pueden ser satisfactorias en gran parte de los casos. Sin embargo, se decide deshabilitar las opciones de hyperthreading, activadas por defecto. Hyperthreading permite que un mismo procesador a nivel de sistema operativo sea interpretado como dos cores virtuales, pudiendo asignar tareas distintas a un mismo procesador físico. Para un sistema final destinado a usuario una aceleración puede ser lograda. En aplicaciones de alto rendimiento como la que se quiere desarrollar podría tornarse perniciosa. Para evitar posibles carreras por los recursos entre los distintos procesos se limitan los 8 cores virtuales a que se muestren como los 4 reales que la unidad de procesamiento posee. A través de las opciones de arranque y la especificación de la afinidad de los procesos se conseguirá de manera manual asegurar que cada una de las utilidades corra en unidades de proceso separadas sin interferir entre ellas.

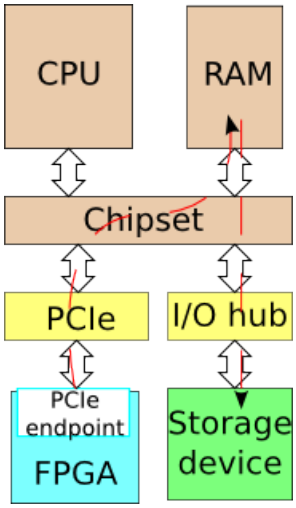
Para continuar con cada uno de los módulos implementados se realiza una explicación descendente, de modo que a partir de la idea general y esquemática se consiga indagar en los detalles más técnicos del capturador realizado.

4.2 Diseño de usuario, capturador

El capturador de tráfico presenta una arquitectura ampliamente conocida como es el modelo productor-consumidor ya presentado en la figura 3.1. El flujo de datos proveniente de la red se monitoriza con el diseño hardware, se propaga mediante transacciones DMA por el bus PCIe a memoria principal desde donde el diseño de usuario será capaz de recopilar la información. La recepción de datos desde el bus PCIe se produce por intervención de un módulo propio del kernel. En la figura 4.2 se muestra la secuencia de los datos.



(a) Esquema de capturador de tráfico basado en FPGA.



(b) Flujo de datos de la aplicación.

Figura 4.2: Diagrama de transmisión de los datos.

En este caso el principal objetivo del diseño del usuario es volcar la información al sistema de almacenamiento. En la sección 5.1 se realiza un estudio sobre el rendimiento de distintos tipos de RAID, por lo que la misión del programa de usuario es realizar un volcado de los datos a un sistema de ficheros XFS soportado por un RAID-0 de 8 discos. El pseudocódigo de la aplicación se muestra en el cuadro 4.2. Sin embargo, hay que tener ciertas consideraciones:

- Para realizar el proceso de manera más eficiente, módulo kernel y diseño de usuario compartirán memoria. Se evita así realizar copias innecesarias con la degradación respectiva del rendimiento.

- El tamaño de bloque transferido (cantidad de datos comunicados en cada ocasión) se desconoce a priori aunque será establecido por el diseño hardware. A nivel de aplicación podría ser un valor dinámico. Debe tenerse en consideración que si se fija una cantidad

inferior a la óptima repercutirá en un aumento de las transferencias DMA a realizar; un límite excesivamente grande puede provocar el desbordamiento de los recursos utilizados. Dado que se utiliza un RAID-0 una buena aproximación es utilizar múltiplos de 8 veces el tamaño del stripe (ocho veces porque se utilizan ocho discos). De esta manera se asegura realizar una escritura óptima en el RAID. Por otra parte que esta cifra sea múltiplo de 512KB asegurará que se utiliza la capacidad máxima de los descriptores DMA. El tamaño de los descriptores y su forma se verá desglosadamente a lo largo de la presente sección.

```
1  Mientras diseño hardware funcionando
2        Thread1                          Thread2
3     Solicita datos          Ubicación, tamaño ← Recibir datos hilo 1
4     Espera  datos           Escribir en disco
5     Comunicar hilo 2 →
6        ubicación y tamaño
7  fin mientras
```

Cuadro 4.2: Algoritmo programa de usuario.

4.2.1 Formato del fichero de captura

Con la exigencia de capturar tráfico a altas tasas de transferencia es necesario buscar un formato de datos que se adecue a las necesidades. En el apartado 3.2 se especifica que el formato debe contener al menos campos relativos al tamaño en bytes de cada una de las tramas capturadas, así como la hora de recepción y contenido del paquete. Volcar los datos sin ningún tipo de contenido adicional puede ser satisfactorio para un grupo concreto de aplicaciones. Sin embargo, en otras ocasiones puede ser que los datos de los paquetes transferidos no contengan toda la información necesaria. Un ejemplo clásico es el realizar un análisis de la distribución de llegada de los paquetes. A su vez, ser capaz de desplazarse por el fichero de captura una vez creado debería ser lo más cómodo posible y por ello se introduce el campo de tamaño de paquete.

Se contempla la posibilidad de volcar los datos en el formato PCAP pero el sobrecoste se hace patente si la tarea del diseño se quisiera aplicar al envío de datos. Si el objetivo fuera leer desde un fichero PCAP, comunicar los datos hacia la tarjeta y ponerlos nuevamente en la red la principal motivación de utilizar PCAP sería su amplio uso más que por motivos de eficiencia. El tener que realizar lecturas de datos a priori innecesarios provoca que las lecturas realizadas sobre el fichero no sean óptimas. En el formato PCAP cada paquete viene precedido de una cabecera con distintos campos informativos que serán desechados por un diseño hardware que exclusivamente busque enviar los datos imitando la tasa de llegada. No se olvide además la presencia de distintas revisiones del formato de captura.

Para ejemplificar esta actividad supongamos la existencia de una traza formada íntegramente por paquetes de 72 bytes. Si el programa de envío leyera de una traza PCAP (en su formato clásico) estaría obligado a leer una cabecera global de 24 bytes. Por cada paquete se lee una cabecera inicial de 16 bytes. Además, como el sistema no puede saber a priori el tamaño de la siguiente trama, realiza una lectura de disco de 72 bytes que es el tamaño que indica la cabecera. A continuación, se repetiría el proceso (obviando la lectura de la cabecera global) hasta terminar con el fichero: leer cabecera local, leer contenido de paquete y transferir información.

El realizar lecturas de tamaño pequeño (1 bloque típicamente en el caso expuesto suponiendo que no hay efectos de caché) no favorece la explotación de los recursos. Mejores aproximaciones se pueden

realizar a este algoritmo, como realizar lecturas de tamaños superiores para posteriormente trabajar sobre la región de memoria donde se albergó el resultado (obsérvese que se añade una copia extra que se preferiría evitar).

Otra opción sería la de tratar los datos en el diseño hardware, en cuyo caso el sobrecoste en el número de bytes transferidos no se puede considerar despreciable. En el caso anteriormente expuesto para una traza formada de paquetes de 72 bytes, el sobrecoste asintótico es del 19 %*, es decir, de cada 100 bytes transferidos aproximadamente 19 son de información ligada a la traza PCAP y desechados parcialmente (timestamp y tamaño siguen siendo útiles).*

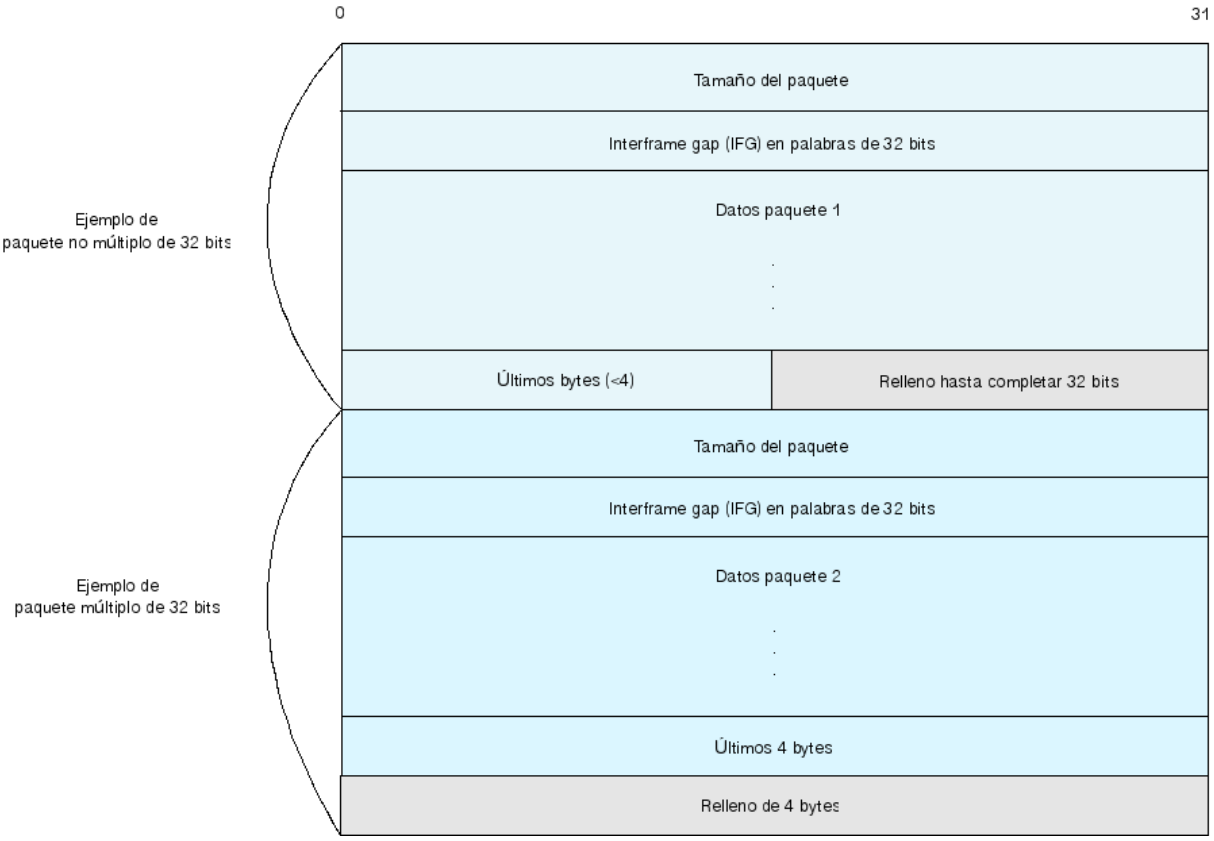


Figura 4.3: Formato propuesto para la captura de datos.

Se plantea el siguiente formato que soluciona las limitaciones expuestas para los paquetes enviados (ver figura 4.3) y simplifica notoriamente el cálculo del tiempo de inactividad entre la transmisión de dos paquetes sucesivos.

- Cabecera local a cada paquete. Incluye dos enteros de 16 bits:
  - Longitud en bytes del paquete.
  - Interframe gap (IFG) expresado en palabras de 32 bits que transcurrirán hasta que se envíe el siguiente paquete en la lista. El mínimo para 10 Gbps [Soc12] son 12 bytes que se traduce en 3 palabras de 32 bits aunque según el caso puede llegarse a reducir a los 9 bytes entre dos paquetes consecutivos. Esta técnica se denomina deficit idle count [Spi12].
- Contenido del paquete.

De aquí en adelante se referirá a este formato como formato simple para evitar posibles confusiones con otros formatos como PCAP. Nótese que la elección del formato de captura está pro-

fundamente influenciada por la posibilidad de reproducir los datos capturados directamente sobre un reproductor basado en una arquitectura similar a la expuesta.

Una consideración añadida que se verá en más detalle a la hora de explicar el desarrollo hardware, es la necesidad de añadir un relleno en el caso de que el tamaño de un paquete en bytes no sea múltiplo de 4. Este relleno consiste en añadir tantos bytes como sean necesarios hasta alcanzar el próximo múltiplo de 4. Si denotamos por m el tamaño del mensaje, el relleno será de

$$4 - m\%4 \text{ bytes}$$

donde % representa el operador módulo. Por simplicidad en la máquina de estados, adicionalmente, si el paquete es múltiplo de 4 bytes, se añade un relleno de 4 bytes. La justificación a este "padding" recae sobre la imposibilidad de comenzar la transmisión de un mensaje en una posición que no sea múltiplo de 32 bits [Soc12]. En el protocolo XGMII se cuenta con un bus de 32 líneas de datos más un total de 4 de control. El tratamiento de los datos como palabras de 32 bits sin necesidad de distinguir posibles casos especiales por la última palabra posibilita que el diseño de reproducción/captura sea más sencillo, agilizando su desarrollo y posibles detecciones de errores. La responsabilidad de dotar del formato apropiado a los datos no recae sobre el diseño de usuario, siendo el propio diseño hardware el que realiza tal acción en tiempo real de captura.

4.3 Driver

La tarea de captura requiere de un driver de altas prestaciones, adecuado al caso concreto que se pretende tratar. En la figura 4.4 se muestran los distintos módulos que son integrados.

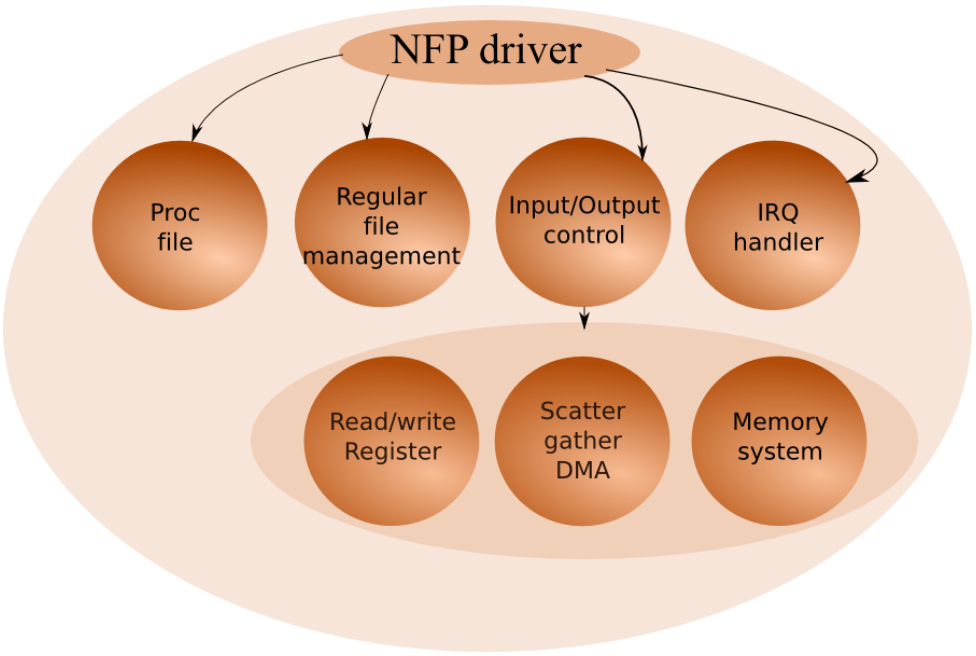


Figura 4.4: Elementos integrados en el driver.

El flujo de instrucciones de un programa software que se ejecuta a nivel de kernel suele ser complejo en comparación con una aplicación de usuario. Se distinguen seis partes bien diferenciadas en la ejecución de un driver:

- Instanciación en memoria. La rutina "init" es ejecutada cuando se carga el módulo en memoria, es decir, el administrador del sistema ejecuta las funciones insmod o modprobe. Generalmente, se inicializan las estructuras necesarias para registrar los dispositivos. En este caso particular, se registra el dispositivo FPGA con vendedor 0x19AA e identificador 0xE004.

- Extracción del módulo. Una vez que el administrador decide dejar de utilizar el driver (invocación de la rutina rmmod), o la estación de trabajo es apagada, la función "exit" es ejecutada. En este instante se pueden liberar los recursos inicializados en la instanciación.

En este punto, aparece una abstracción que para un dispositivo como es una FPGA no es demasiado significativa. Sin embargo, para otra clase de periféricos, como pudiera ser una memoria flash extraíble, se puede interpretar de manera más sencilla. En el momento de la inicialización, se registra una lista de dispositivos compatibles y se definen las instrucciones a ejecutar cuando alguno de ellos sea conectado,suspendido, expulsado, etc. De esta manera, un driver puede manejar un dispositivo desde el instante en el que es conectado y hasta su desconexión. Sin necesidad de reinstanciar el módulo una nueva unidad que pudiera ser insertada y coincidiera con el patrón de dispositivos soportados por el driver podría ser gestionada.

Siempre que un driver maneje un dispositivo tendrá que informar, al menos, de las acciones "probe" y "remove" (registro y desconexión) aunque en el caso de una FPGA el dispositivo siempre esté conectado. Es decir, tras haber ejecutado la rutina "init" el flujo de datos es sucedido por:

- Registro del dispositivo. En la rutina "probe" se reserva la memoria necesaria por el programa: semáforos, colas de trabajo ("workqueues"), "spinlocks", métodos de comunicación del usuario: dispositivos de caracteres y ficheros virtuales.

- Expulsión del dispositivo. La función "remove" deshace todas las operaciones realizadas a lo largo de la rutina "probe" y las posibles modificaciones que pudiera haber causado la interacción con el usuario. Tras el registro y expulsión de un dispositivo, el estado del mismo debe ser igual al que ofrecía antes de la conexión. Es decir, no se debe alterar su configuración salvo que sea la propia finalidad (por ejemplo, la escritura en un medio de almacenamiento, los datos deben preservarse después de la extracción).

En el cuadro 4.3 se muestra como se declaran las funciones "init" y "remove". En el cuadro 4.4 se muestra una versión simplificada de las rutinas para el registro de la FPGA por el módulo, rutinas "probe" y "remove".

```
1  static struct pci_device_id pci_id[] = {
2      {PCI_DEVICE(PCI_VENDOR_ID_NFP, PCI_DEVICE_ID_NFP)}, /*
           PCI_VENDOR_ID_NFP y PCI_DEVICE_ID_NFP son enteros de 16 bits */
3      {0}
4  };
5
6  MODULE_DEVICE_TABLE(pci, pci_id);
7
8  static struct pci_driver pci_driver = {
9    .name = ''nfp'',
10   .id_table = pci_id,
11   .probe = nfp_probe,
12   .remove = __devexit_p(nfp_remove),
13 };
14
```

```
15  static int __init nfp_init(void)
16  {
17    return pci_register_driver(&pci_driver);
18  }
19
20  static void __exit nfp_exit(void)
21  {
22    pci_unregister_driver(&pci_driver);
23  }
24
25  module_init(nfp_init);
26  module_exit(nfp_exit);
```

Cuadro 4.3: Instanciación de un módulo kernel.

```
1   static int __devinit nfp_probe(struct pci_dev *pdev, const struct
        pci_device_id *id){
2     pci_enable_device(pdev);
3     pci_set_dma_mask(pdev, DMA_BIT_MASK(64));
4     pci_set_drvdata(pdev, card);
5
6     if (pci_resource_flags(pdev, 0) & IORESOURCE_MEM) {
7           pci_request_regions(pdev, "nfp");
8           card->bar0 = pci_iomap(pdev, 0, pci_resource_len(pdev, 0));
9           nfpiface_probe(pdev, card);
10    }
11    nfpioctl_probe(pdev, card);
12    nfpproc_probe(pdev, card);
13  }
14  static void __devexit nfp_remove (struct pci_dev *pdev)
15  {
16
17      nfpproc_remove (pdev, card);
18      nfpioctl_remove (pdev, card);
19
20      nfpiface_remove (pdev, card);
21      pci_iounmap (pdev, card->bar0);
22
23      pci_release_regions (pdev);
24      pci_set_drvdata (pdev, NULL);
25      pci_clear_master (pdev);
26      pci_disable_device (pdev);
27    }
28  }
```

Cuadro 4.4: Función de registro y expulsión de un dispositivo.

Una vez ejecutadas las rutinas de “init” y “probe”, el driver se suspende (no vuelve a ejecutarse) hasta que ocurra alguno de los siguientes supuestos:

- Ejecución de las rutinas asociadas a la interacción con el usuario, ver apéndice A.
- Manejo de interrupciones (analizado nuevamente en el apéndice A).

La única posibilidad de ejecución del driver es a través de alguno de los dos casos anteriores hasta que sea desinstanciado. El manejo de una interrupción desemboca por la terminación

de una operación de DMA por lo que podría considerarse un subconjunto de las rutinas asociadas a la interacción con el usuario. Una vez cargado el driver y registrado el dispositivo, las siguientes características están presentes para la captura de tráfico y comunicación con el usuario:

- Fichero virtual "/proc/nfp/nfp_report" para la medida de prestaciones, unidad "proc file" en la figura 4.4. Es un fichero de lectura exclusivamente y proporciona información sobre la actividad del motor DMA en el último segundo (tiempo activo, tiempo esperando a datos y bytes transferidos), así como datos ligados al diseño de captura: estados, ocupaciones de las FIFOs intermedias. Además, indica si hay una posible pérdida de datos si no se está alcanzando la tasa objetivo.

  Es importante destacar, que en las versiones del kernel superiores a la 3.10 se han visto modificados los prototipos de las funciones encargadas de registrar un fichero virtual. El código del módulo está adaptado para poderse ejecutar en un equipo con cualquiera de las dos configuraciones (versiones 2.6.32 hasta 3.13) sin necesidad de realizar ninguna alteración adicional. En el apéndice A se muestran las diferencias entre ambas versiones.

- Manejo de "huge pages", unidad "memory system" en la figura 4.4. Para el soporte de páginas de tamaño mayor a 4 KB se requiere que el usuario reserve dicha región de memoria y lo comunique al módulo mediante un dispositivo de caracteres. En la sección 4.3.1 se describe el proceso.

- Escritura en fichero, unidad "regular file management". Se ofrece la posibilidad de que sea el propio módulo kernel el encargado de escribir en disco sin necesidad de comunicar los datos al usuario.

- Transferencias desde el anfitrión a la placa y de la placa al anfitrión mediante operaciones DMA "scatter-gather". La posibilidad de referenciar grandes regiones de memoria, junto con el hecho de que se implemente la posibilidad de realizar múltiples operaciones de DMA, favorece que la CPU permanezca inactiva la mayor parte del tiempo.

- El diseño de usuario puede seleccionar una transferencia DMA para que genere una interrupción cuando sea finalizada. El flujo puede verse interrumpido, en cualquier momento, para atender una petición. Dado que hay que realizar una serie de operaciones que involucran el acceso a regiones de memoria no necesariamente consecutivas y podrían demorar el proceso, se opta por encolar el trabajo para realizarlo en momentos de menor ocupación de CPU. Las operaciones a realizar en una transferencia de la placa al anfitrión es el recuento de los bytes transferidos (a través de la información actualizada en los descriptores) y la actualización de los índices internos.

- Dispositivo de caracteres "/dev/nfp", unidad "input/output control" en la figura 4.4. Soporta varias operaciones input/output control (IOCTL). Se detallan en la siguiente lista y en el apéndice A se muestra parte del código de la implementación:

  – NFPIOC_REGISTER_BUFFER. El diseño de usuario durante la inicialización reserva la memoria donde se albergarán los datos. Esta memoria es compartida por el driver y el diseño de usuario. A través de la invocación de esta operación el módulo es informado de la ubicación de memoria (y tamaño) sobre la que realizarán las operaciones.
  – NFPIOC_UNREGISTER_BUFFER. Comunica al módulo que la región de memoria de la que estaba haciendo uso (la región concedida mediante la llamada IOCTL

NFPIOC_REGISTER_BUFFER) debe ser liberada. Tras su llamamiento el diseño de usuario está en el derecho de liberar la región de memoria o utilizarla para otros fines no compartidos con el driver.

- NFPIOC_S2C_DMA. Solicita realizar una operación system to card (S2C) mediante DMA. A través de los argumentos se indica el offset respecto a la región de memoria indicada con la llamada a NFPIOC_REGISTER_BUFFER de donde se debe leer y el tamaño a transferir.
- NFPIOC_C2S_DMA. Solicita realizar una operación card to system (C2S) mediante DMA. A través de los argumentos se indica el offset respecto a la región de memoria indicada con la llamada a NFPIOC_REGISTER_BUFFER donde se puede escribir y el tamaño máximo reservado.
- NFPIOC_READ_32. Realiza una lectura de una palabra de 4 bytes en un offset especificado del BAR indicado.
- NFPIOC_WRITE_32. Realiza una escritura de una palabra de 4 bytes en un offset especificado del BAR indicado.

```
1  Instancia:
2    Solicitar recurso con VENDOR=0x19AA y DEVICE=0xE004.
3    Pedir memoria y recursos sistema (semáforos y colas de trabajo).
4    Habilitar PCIe.
5    Establecer dispositivo como maestro.
6    Realizar el map de los distintos base address register.
7    Habilitar char device y crear estructura bajo /proc.
8    Habilitar interrupciones.
9  Ejecución:
10   Si hay IOCTL:
11     Si registrar buffer:
12         Convertir la dirección virtual a dirección de bus. Guardar la
               dirección de inicio de cada página.
13     fin si
14     Si liberar buffer:
15         Esperar a terminar las operaciones activas.
16         Olvidar las direcciones de bus.
17     fin si
18     Si operación C2S:
19         Configurar descriptores en dirección hacia el anfitrión.
20         Actualizar registros FPGA.
21         Habilitar interrupción en el último descriptor.
22
23         Dormir en semáforo hasta que la rutina manejadora nos despierte.
24         Actualizar datos sobre tamaño recibidos.
25     fin si
26     Si operación S2C:
27         Configurar descriptores en dirección hacia la tarjeta.
28         Habilitar interrupción en el último descriptor si el usuario lo
               indica.
29         Actualizar registros FPGA.
30
31         Dormir en semáforo hasta que la función manejadora nos despierte
               (sólo si el usuario lo indica).
32     fin si
33   fin si
```

Cuadro 4.5: Descripción en lenguaje natural del módulo kernel.

En líneas generales el pseudocódigo del driver queda reflejado en el cuadro 4.5 y la rutina de la función manejadora en 4.6.

```
1  Función manejadora de interrupciones:
2      engine ← Obtener motor que ha causado la IRQ.
3      up( semáforo[engine] )
```

Cuadro 4.6: Pseudocódigo función manejadora.

De modo que el diseño de usuario (a través de una capa intermedia) deberá realizar un "map" de la región de memoria asociada a las "huge pages" y comunicar la dirección a través de la IOCTL NFPIOC_REGISTER_BUFFER. En el momento que se quiera empezar a recibir datos del diseño hardware se deberá solicitar a través de la llamada a la IOCTL NFPIOC_C2S_DMA. Este proceso se repite indefinidamente hasta que el usuario no necesite de más datos.

Esta pequeña visión global del diseño a nivel software hace más sencillo profundizar en el tema del manejo de la memoria y las transferencias DMA.

4.3.1 Compartición de memoria entre el nivel de usuario y sistema operativo

El uso de páginas de tamaño no estándar (páginas de tamaño superior a 4KB tradicionalmente), no ha sido inicialmente previsto para su uso a nivel de kernel en un entorno linux. Su aplicación en herramientas que se ven obligadas a usar una cantidad masiva de datos, como pudiera ser un gestor de base de datos, es algo ampliamente desarrollado y estudiado mas no tanto su empleo desde un módulo ([Wha13] o [Ora14]).

Sin embargo, dado que para el uso intensivo de operaciones que se pretende realizar, que involucran el trabajo con memoria, el acceso a regiones contiguas contribuiría con un mayor rendimiento. Se evalúa la implantación de páginas no estándar en el desarrollo del driver. Aparece el inconveniente de la disminución del grado de disociación entre módulo y nivel de usuario, ya que el programa de usuario será quien reserve estas páginas (a través de una capa intermedia, "middleware") y el diseño del driver está obligado a conocer que la actividad será, efectivamente, de esta manera. En cualquier caso, antes de poder utilizar esta región de memoria reservada en el arranque (ver 4.1.3) es necesario montar el sistema de ficheros como se indica en el cuadro 4.7.

```
1  mkdir -p /mnt/huge
2  mount -t hugetlbfs none /mnt/huge
```

Cuadro 4.7: Montaje del sistema de ficheros hugetlbfs.

Con el fin de acceder a estas regiones de memoria, a nivel implementatorio, se realiza un map de tamaño igual al número total de páginas por el tamaño de cada página sobre el archivo en el que se ha montado el sistema de ficheros "hugetlbfs" (siguiendo las instrucciones del cuadro 4.7 el archivo es "/mnt/huge"). El sistema operativo se encargará de devolver un puntero a una región de memoria virtual donde todas las páginas aparecerán en posiciones sucesivas. Este puntero se comunica al módulo propio para la obtención de las direcciones de bus de las páginas.

En la figura 4.5 se muestra un esquema del proceso que toma lugar y en el cuadro 4.8 el comando mmap correspondiente para la reserva de memoria. Como ya se ha mencionado el diseño del módulo capturador es capaz de obtener las direcciones de bus de cada una de las páginas sin necesidad de que nueva información se vuelva a comunicar a lo largo de todo el

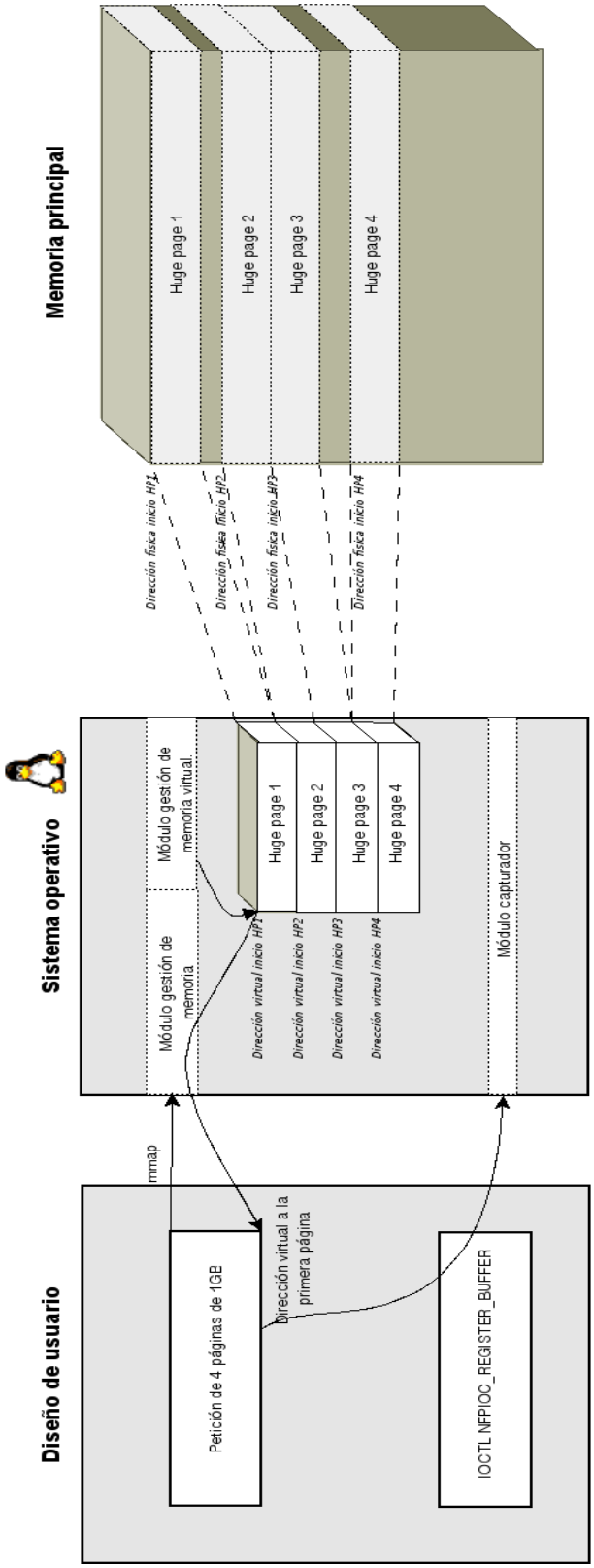


Figura 4.5: Proceso de solicitud de huge pages y su comunicación al driver.

proceso. Sin en cambio, si será necesario asegurar mecanismos de comunicación entre driver y

diseño de usuario para evitar que se accedan a datos no disponibles o se sobrescriban regiones aún no procesadas de memoria.

```
1       file = open("/mnt/huge", O_CREAT | O_RDWR, 0755);
2       void *data = mmap( NULL, NUMBER_HUGEPAGES*HUGEPAGE_SIZE,
3                          PROT_READ | PROT_WRITE,
4                          MAP_SHARED, file, 0 );
```

Cuadro 4.8: Mmap para la solicitud de la memoria de cuatro huge pages a nivel de usuario.

Una vez obtenida la dirección virtual basta con realizar la llamada IOCTL expuesta en el cuadro 4.9.

```
1       struct dma_transfer di;
2       FILE *fd;
3
4       fd = open( "/dev/nfp", O_RDWR);
5
6       di.data   = data; /* Datos obtenidos previamente en la llamada a
            mmap. */
7       di.length = NUMBER_HUGEPAGES*HUGEPAGE_SIZE;
8
9       ioctl(fd, NFPIOC_REGISTER_BUFFER, &di);
```

Cuadro 4.9: Mmap para la solicitud de la memoria de las huge pages a nivel de usuario.

Como mejoras futuras en el diseño, se podría simplificar notoriamente mediante los siguientes cambios que contribuirán en la legibilidad del código y aislamiento entre capas (driver y diseño de usuario):

- Implementación de un driver en el nivel de usuario, módulo UIO. De esta forma se posee acceso completo a las huge pages desde el propio driver sin mediación del programa de usuario. Esta alternativa es la optada por otros recursos como Intel DPDK [Int].
- Alternativamente, se puede desarrollar una librería que permita el manejo y petición de páginas de tamaño no estándar desde una aplicación ejecutándose en modo kernel.

4.3.2 Configuración del motor DMA y descriptores de memoria

Este punto es altamente dependiente del diseño hardware implementado. Es necesario que se establezca una interfaz común entre ambas plataformas. Por motivos de equilibrio entre tiempo de desarrollo y coste, se elige la opción de reutilizar un IP core desarrollado por Northwest Logic [Log]. El core presenta una serie de características destacables para el producto:

- Soporte para PCIe hasta su tercera generación.
- Múltiples motores DMA de alto rendimiento.
- Soporte para memorias estándar memoria de acceso aleatorio duals (DRAMs).
- Soporte para la interfaz (AXI4-Stream).

- Soporte para interrupciones MSI, MSI-X y legacy.
- Certificado por PCI-SIG.

El diagrama de bloques del diseño aparece en la figura 4.15. La incorporación del core simplificará en gran medida el diseño que se ejecutará en la FPGA, debido a que la comunicación con el host se limitará a la negociación con un bus AXI4-Stream sin necesidad de entrar en detalles más intrínsecos del bus PCIe (en [PS05] se detalla el protocolo de comunicación basado en paquetes usado por PCIe).

La configuración de una transacción de memoria (copia o escritura) involucra dos elementos: actualización de la información en alguno de los motores DMA y la creación de, al menos, una estructura descriptor que se comunicará al motor.

- Una estructura del tipo descriptor contiene la información básica de una transacción. Se incluye la dirección de bus de la región de memoria donde se desea leer/escribir y el tamaño de dicha región como campos básicos. Sin embargo, el conjunto completo de campos es más amplio lo que permite diseñar aplicaciones más elaboradas que hagan un uso reducido de CPU:
  - Flags de control: flag de inicio de paquete. Indica si el descriptor contiene el inicio de un paquete.
  - Flags de control: flag de fin de paquete. Indica si el descriptor contiene el final de un paquete. No lleve a confusión el término paquete con traza de red. En este caso paquete se considera una unidad de envío. A nivel de bus AXI4-Stream puede interpretarse como la activación de la señal LAST (ver [ARM10] para la especificación completa del protocolo o el resumen realizado en el apéndice B).
  - Flags de control: interrupción en ejecución exitosa. Activa la interrupción MSI/MSIX pertinente si el motor de DMA también tiene activada dicha opción y la operación de transmisión finaliza correctamente.
  - Flags de control: interrupción en ejecución errónea. Activa la interrupción MSI/MSIX pertinente si el motor de DMA también tiene activada dicha opción y la operación de transmisión finaliza de manera inesperada. Se actualizan los campos del descriptor con información sobre el error (ver [Log] para más detalles).
  - Dirección de bus del siguiente descriptor.
  - Dirección de bus de la región de memoria sobre la que transferir.
  - Tamaño disponible. El campo tiene una longitud de 20 bits en el diseño. Se establece como unidad de envío por cada descriptor el valor $2^{19} = 512$ KB. Nótese que el valor máximo representable sería $2^{19} + 2^{18} + \ldots + 2^1 + 2^0 = 2^{20} - 1$ que no es potencia de 2, hecho que podría tener influencia negativa en términos de cálculos aritméticos que involucren dicho factor y por eso se opta por tomar 2^{19} como el tamaño estándar de transferencia.
  - Bytes escritos. Aunque el descriptor apunte a una región de 2^{19} bytes, puede darse el caso de que la información leída/escrita en dicha región sea inferior. Este campo indica la cantidad real de datos escrita.

  Merece la pena destacar el campo cuyo contenido es la dirección de otro descriptor. El motor DMA que procese el descriptor tras completar la operación se encargará de ir a buscar el siguiente sin necesidad de mediación de la CPU. Esto permite crear listas de

descriptores capaces de transferir una gran cantidad de datos, en regiones no necesariamente contiguas y sin un procesamiento adicional por parte del anfitrión. En la imagen 4.6 se esquematiza el concepto.

- A su vez, el core dispone de múltiples motores de DMA. Cada uno de ellos contiene información común a un conjunto de operaciones. Es importante notar que cada motor se configura a través del BAR0 y unos desplazamientos fijos (para detalles más intrínsecos [Log]). Entre las opciones disponibles, se permite seleccionar si tras la finalización de una copia se generarán interrupciones (su inhabilitación provocará que las preferencias de un descriptor sean ignoradas) o la dirección de bus de la estructura descriptor que se tomará para su próxima operación si en el momento anterior el motor estaba inoperativo. La lista completa de campos que un motor posee es:
  - Flag de estado: dirección del motor. Indica si se encarga de realizar transferencias desde la placa hacia el host o en la dirección inversa.
  - Flag de estado: tamaño máximo de los descriptores soportados. El número de bits máximo que se podrán transferir es de 2 elevado a este valor.
  - Flag de control: interrupciones activas. Indica si las interrupciones por error/completitud serán generadas.
  - Flag de estado: descriptor completado. Indica si se ha completado un descriptor. El valor se puede sobrescribir para reiniciar el flag. Útil para realizar una implementación basada en polling.
  - Flag de estado: error en la alineación del descriptor. El descriptor no estaba localizado en una posición de memoria múltiplo de 256 bits.
  - Flag de estado: error por detención software. Se ha comunicado la cancelación de una operación por parte del diseño de usuario.
  - Flag de estado: alcanzado el final de la cadena. Se ha seguido la cadena de descriptores hasta alcanzar un descriptor sin sucesor (dirección de bus del siguiente descriptor igual a 0x0).
  - Flag de estado: el motor está funcionando. Hay alguna operación en curso (podría darse el caso de que estuviera detenida en una cadena de descriptores).
  - Flag de control: petición de reinicio. Indica que se cancele la actual operación en curso y se detenga el motor.
  - Flag de control: petición de inicio. Indica que se comience la operación indicada por el siguiente descriptor.
  - Puntero al siguiente descriptor. Cuando el motor está parado, la escritura en este registro indica el siguiente descriptor a procesar.
  - Puntero al siguiente descriptor que posee el software. En el procesamiento de una cadena, si el descriptor a procesar coincide con este registro, la cadena se detiene hasta la actualización de este elemento.
  - Puntero al último descriptor completado.
  - Tiempo esperando a la actualización del último descriptor que posee el software (hay una cadena de descriptores en curso pero el siguiente descriptor coincide con una región en posesión del software).
  - Número de bytes transferidos en el último segundo.

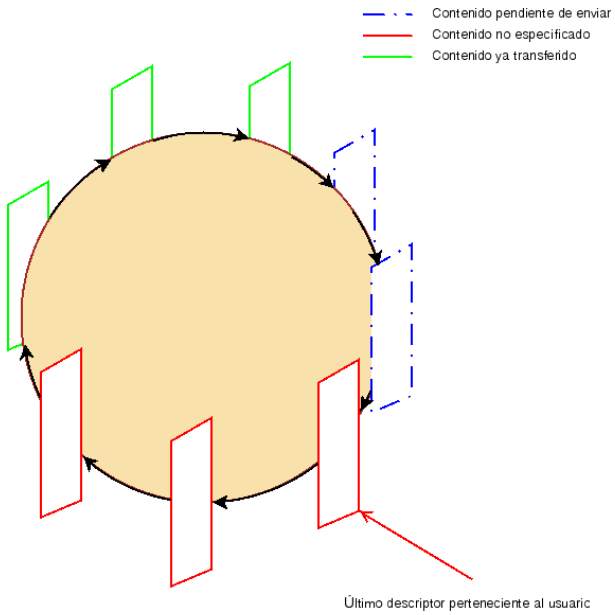


Figura 4.6: Representación esquemática de un anillo de descriptores en transferencia.

Un campo de vital importancia es la dirección de bus de la última estructura del tipo descriptor que pertenece al usuario.

Los motores de DMA son capaces de detener la ejecución de copia de datos en un punto especificado (haciendo uso del campo último descriptor que pertenece al usuario), favorece que se pueda configurar inicialmente un anillo de descriptores, que apunten a una región fija de memoria cada uno de ellos individualmente, debiendo realizar unas modificaciones básicas (como la activación/desactivación del bit de generación de interrupciones si fuera preciso, bytes válidos dentro de la región apuntada, etc.) para el envío/recepción de nuevos datos. En la figura 4.6 se muestra como a medida que se capturan datos, se pueden transferir automáticamente a la región apuntada por los descriptores a menos que el programa a nivel de usuario no los haya procesado. En tal caso el sistema se detiene a la espera de que dicho evento tenga lugar (actualización del campo último descriptor perteneciente al usuario en los registros del diseño hardware).

Es decir, inicialmente se reservan e inicializan tantos descriptores como sean necesarios para "mapear" la región de memoria que ofrecen las huge pages. Cada uno de ellos apunta al inmediatamente siguiente y se corresponde con la región de memoria:

$$(\textit{Número descriptor}-1)\times \textit{Tamaño de la región apuntada por descriptor}$$

(ver imagen 4.7). El último descriptor apunta al primero y sobre este último recae el indicador de descriptor en propiedad del diseño de usuario. Una vez que se haya transferido tanta información como para haber alcanzado el descriptor final, se espera que el diseño de usuario comunique que la región de memoria está disponible. Este proceso se repite indefinidamente hasta haber procesado la cantidad de información requerida por el experimento.

Para los diseños realizados no existe la necesidad de utilizar más de un motor DMA. Por tanto, a pesar de existir un total de 3 motores de DMA en cada dirección (desde la tarjeta hacia el anfitrión y viceversa) y ser capaces de idear aplicaciones que reciban y envíen datos simultáneamente, por ejemplo, no serán objeto de estudio en el presente documento. El uso de motores para el envío/recepción de datos se tratará de manera independiente y no se probará su funcionamiento en paralelo.

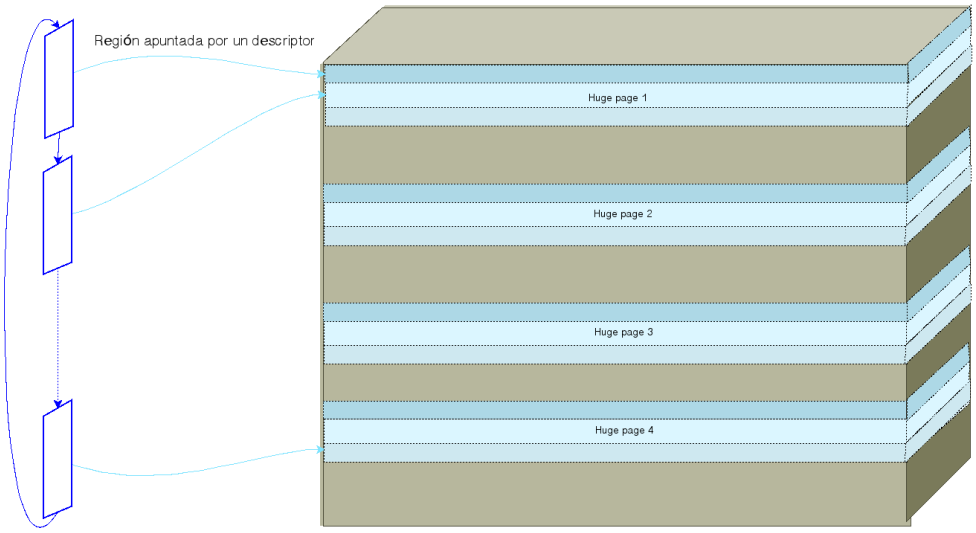


Figura 4.7: Asociación entre memoria principal y descriptores.

4.4 Desarrollo hardware

En esta sección se desglosa las tecnologías en uso por el desarrollo que será ejecutado sobre la FPGA, así como las cuestiones de diseño y elementos de terceros utilizados. Se comienza la sección especificando el entorno de desarrollo para continuar con la capa física del modelo Open Systems Interconnection model (OSI) y el protocolo PCIe.

4.4.1 Protocolos independientes de la interfaz a 10 Gbps: XGMII y XAUI

Dado el bajo nivel al que se trata la información en esta aplicación, un conocimiento mínimo sobre las distintas capas y protocolos utilizados en el nivel 1 del modelo OSI, la capa física, para transferencia de datos a 10 Gbps es requerido. Dado que el capturador de tráfico omite la implementación del resto de capas del modelo, y para una comprensión mayor del diseño hardware desarrollado, en este punto se explica la arquitectura del componente de capa física.

En la capa física del modelo OSI (PHY), se distinguen tres elementos. La capa dependiente del medio físico, Physical Media Dependent (PMD), la subcapa Physical Medium Attachment Sublayer (PMA), encargada de la sincronización y detección de octetos, y la subcapa de codificación física, Physical Coding Sublayer (PCS). A nivel de definición del nivel 1, el IEEE 802.3 [Soc12] define dos modelos: LAN PHY y WAN PHY. WAN PHY extiende las características de las funcionalidades de la capa para LAN y se distinguen exclusivamente por el PCS.

Entre la capa de enlace,media access control (MAC), y la capa física se encuentra el protocolo XGMII, 10 Gigabit Media Independent Interface. Provee de operaciones full duplex a la tasa de 10 Gbps entre MAC y PHY. Cada dirección es independiente y contiene un bus de 32 bits de datos para cada dirección, al igual que un reloj y señales de control. En total la interfaz tiene un ancho de 74 bits.

Mientras que XGMII proporciona pipeline a 10 Gbps, la transmisión independiente de la señal de reloj y de datos, junto con el requisito de transmitir datos tanto en los flancos ascendentes como descendentes de reloj, es difícil asegurar el correcto enrutamiento más allá de la corta distancia de 7 cm. Por esta razón, aplicaciones chip a módulo óptico (entre otras) no son prácticas con esta interfaz.

Para solucionar estos inconvenientes se desarrolló la interfaz XAUI como extensión a XGMII. XAUI es una interfaz full duplex que utiliza cuatro enlaces diferenciales en cada dirección

para alcanzar 10 Gbps de transferencia de datos. Cada enlace funciona a 3.125 Gbps para dar cabida a los datos y la sobrecarga asociada a la codificación 8B/10B. Se extiende el alcance funcional del protocolo XGMII a unos 50 cm. La conversión entre las interfaces XGMII y XAUI ocurre en el XGXS (XAUI Extender Sublayer). En la figura 4.8 quedan esquematizados todos estos conceptos. Para una referencia mayor sobre el asunto se recomienda consultar [For00].

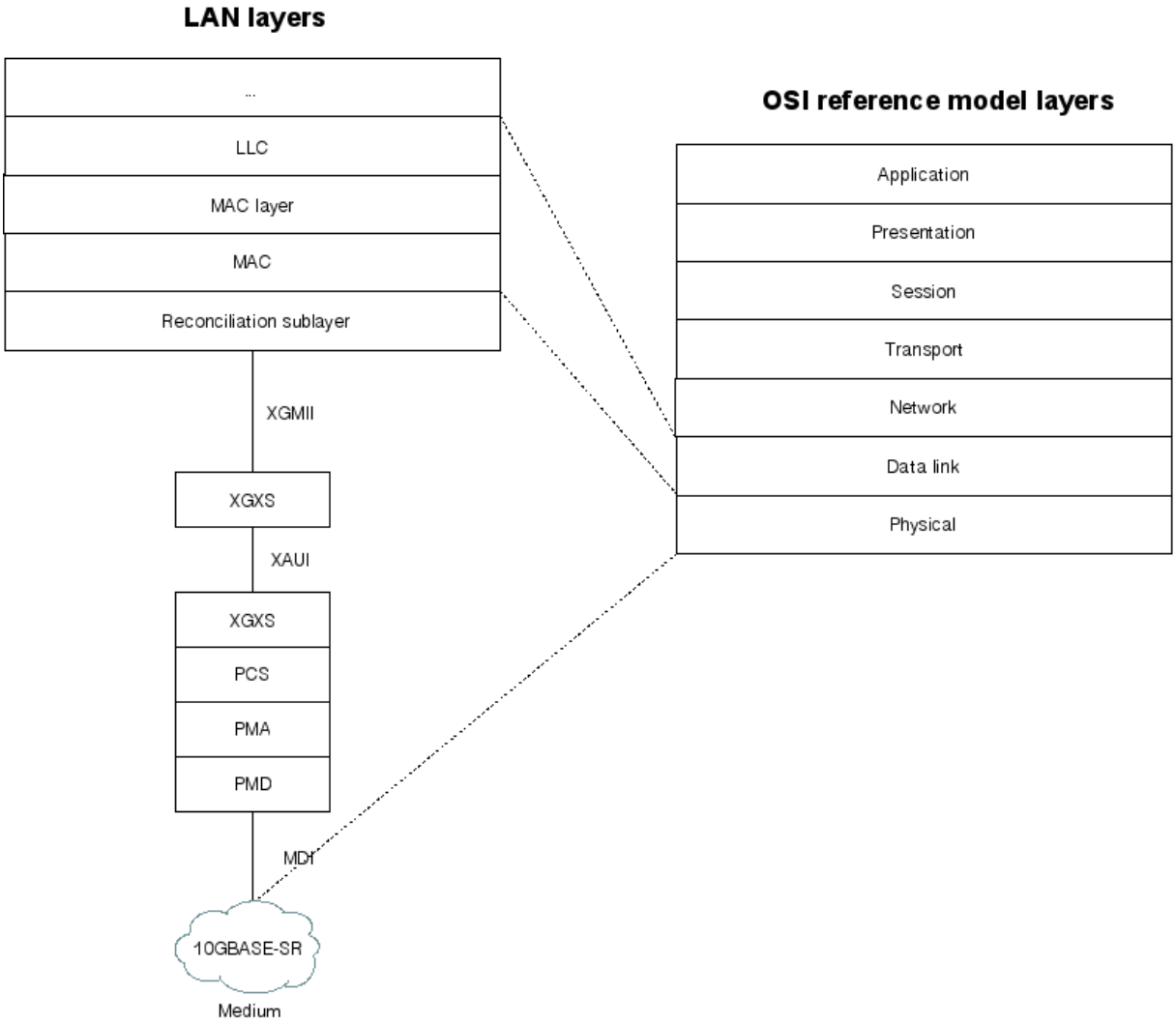


Figura 4.8: Protocolo OSI. Desglose capa de nivel físico.

En la figura 4.9 se muestra la necesidad de implementar parte de la capa XGXS en el transceptor y, de manera independiente, para realizar la traducción al formato XGMII, antes de la subcapa de reconciliación (RS).Se hace especial hincapié en justificar este diseño debido a las limitaciones de rango del protocolo XGMII. En el apéndice B se explica como realizar el map de las señales del transceptor en el diseño.

A nivel de capturador se decide monitorizar las señales en el nivel de la subcapa de reconciliación sin llegar a implementar ningún nivel superior a éste. Es decir, se monitorizan las señales del protocolo XGMII y se guardan los datos obtenidos de este protocolo con las ventajas e inconvenientes que pudiera acarrear. Un inconveniente, según el uso que se desee dar al capturador, es la captura de paquetes con errores en el CRC ethernet sin percatarse e informar de dicha anomalía. Sin embargo, dada esta decisión de diseño, el overhead de utilización de CPU detectado en 2.2 será paliado tanto en su parte asociada al procesamiento de la pila TCP/IP

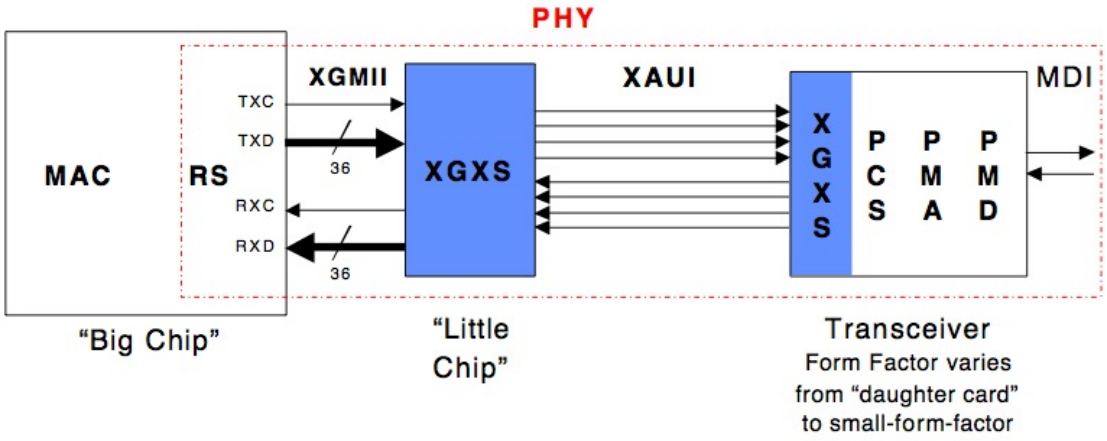


Figura 4.9: Ejemplo de implementación de XGXS.

por parte del sistema operativo, como en su parte debida al procesamiento de la capa de enlace (ya que no habrá ninguna demora extra al no realizar tales acciones).

Para la implementación de la conversión entre XGMII y XAUI se utiliza el LogiCORE IP XAUI v10.4 [Cor12c], encargado de encapsular dicha funcionalidad. En [Cor12c] se propone utilizar el siguiente modelo de la figura 4.10. El diseño realizado será muy similar al propuesto pero sin utilizar la unidad de Management Data Input/Output (MDIO) de este core.

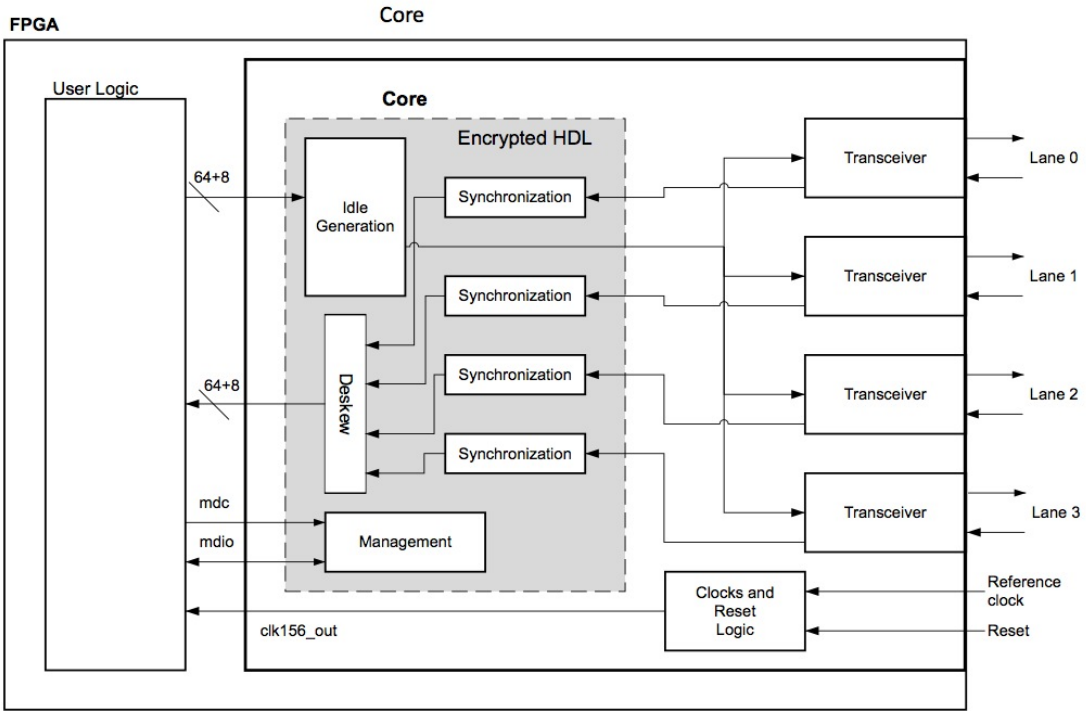


Figura 4.10: Diseño basado en XGMII sugerido por [Cor12c].

Nótese la analogía en la imagen 4.10 con la versión propuesta en la figura 4.2a. Una observación a realizar sobre la imagen es la entrada de un bus de 72 bits de datos (64 bits de datos + 8 bits de control) y un bus de salida con el mismo ancho. Estos buses se corresponden con las

señales RXC, RXD, TXC, TXD del protocolo XGMII (ver, por ejemplo, figura 4.9). Que tengan un ancho del doble se debe a que el protocolo XAUI transmite datos tanto en flanco ascendente como descendente de reloj. El core utilizado agrega dichos datos para transmitirlos en flanco de subida.

Establecido el punto de escucha, es necesario identificar las señales de control en el protocolo [Soc12]. El flujo de datos queda definido en el cuadro 4.10.

```
1  <Inter-Frame><Preamble><sfd><Data><efd>
```

Cuadro 4.10: Flujo de datos en el protocolo XGMII.

Donde cada uno de los elementos representa:

- "Inter-frame". Periodo de inactividad entre el envío/recepción de dos paquetes consecutivos. Se caracteriza por presentar cada octeto de datos el valor 8'h07.
- "Preamble". Comienza la transmisión de un paquete. Consiste de 7 octetos con el siguiente valor en hexadecimal: 56'h555555555555FB. En la primera oscilación del reloj la línea 3 toma el valor 55, al igual que la línea 2 y 1. La línea 0 toma el valor 8'hFB. En el siguiente flanco línea 0,1 y 2 a 8'h55, mientras que la línea 3 tiene el valor del start frame delimiter como se muestra en la figura 4.11. Nótese que los primeros bytes transferidos son los de la primera línea y los bits más significativos se escriben más a la derecha dentro de la misma línea. Así, por ejemplo, la secuencia 1010 se corresponde con el valor 5 en hexadecimal o el octeto 10101011 se corresponde con el byte D5. Solamente esta permitido el valor 8'hFB en la línea 0 del transceptor. En el caso del core usado, byte 0 u 4.

Lane 0	Lane 1	Lane 2	Lane 3
Start	10101010	10101010	10101010
10101010	10101010	10101010	10101011

Figura 4.11: Preámbulo más delimitador de inicio para transmisión en el protocolo XGMII.

- "Start Frame Delimiter (sfd)". Sigue inmediatamente al preámbulo y consiste en el octeto marcado por 8'hD5 en el envío de un paquete. En recepción la capa de reconciliación convierte este byte en el octeto de comienzo: 8'h55.
- "Data". Los datos del paquete vistos como un flujo de octetos.
- "End frame delimiter (efd)". Indica el posible código de terminación de un paquete. 8'hFE indica que se ha completado la transferencia con un error, 8'hFD indica que se ha completado la transferencia del paquete exitosamente.

En la tabla 4.1 aparecen los caracteres especiales junto con las señales de control asociadas para la interfaz XGMII implementada en el core. Se resto de valores de la señal del preámbulo llevan asociados el bit de control a 1'b0.

El "map" entre los bit de las líneas de control y los bytes de datos quedan contemplados en la tabla 4.2.

Por tanto, para la captura de un paquete bastará el desarrollo de un módulo a la escucha del caracteres de control 8'hFB. Una vez reconocido el patrón se deben guardar todos los octetos hasta detectar el patrón 8'hFE o 8'hFD. En ambas, situaciones se presupone que la secuencia

Datos (Hex)	Control	Nombre
00-FF	0	Datos
07	1	Idle, inactivo
FB	1	Comienzo
FD	1	Finalización
FE	1	Error

Tabla 4.1: Especificaciones caracteres de control en el protocolo XGMII.

Bit de control	Bit en el bus de datos
0	7:0
1	15:8
2	23:16
3	31:24
4	39:32
5	47:40
6	55:48
7	63:56

Tabla 4.2: Relación entre el byte de control y el bus de datos.

es válida únicamente si el bit de control asociado está activo. En la figura 4.12 se muestra una sencilla máquina de estados que modeliza el comportamiento a implementar.

Esta máquina de estados incluye todas las transacciones pero no incluye la funcionalidad al completo del diseño por simplicidad. Es necesario que además se implemente un contador de bytes y se tenga en consideración el interframe gap entre dos paquetes consecutivos. El módulo encargado de realizar la conversión entre el protocolo XGMII y AXI4-Stream (más información en el apéndice B) se denotará como xgmii2axi y el diagrama de bloques se muestra en la figura 4.13.

De este primer módulo cabe destacar la presencia de dos registros de 128 bits, control y status. Permiten configurar aspectos tales como el número total de paquetes a capturar, línea a monitorizar dentro de las 4 interfaces disponibles, comunicar un reinicio de la aplicación, etc. En el apartado 4.5.1 se detalla en mayor profundidad cada uno de los campos.

Como salidas del módulo hay dos interfaces AXI4-Stream que se corresponden con las cabeceras y los datos de cada trama ethernet escuchada. La cabecera tiene los campos de la sección 4.2.1, es decir, cabecera e interframe gap. Por cada paquete recibido una palabra de 64 bits será generada por la interfaz AXI con el prefijo y tantas palabras como sea necesario para comunicar los datos del paquete. Si un paquete no es múltiplo de 8B, al leer el último byte del paquete es posible que se reciba información de la siguiente trama. Este comportamiento no se evita en este módulo y debe verificarse en capas superiores (módulo concat).

Este módulo de concatenación, tiene por objetivo leer una cabecera y a continuación tantos bytes de datos como se indiquen en ella. Si se leen datos pertenecientes al siguiente paquete se tiene en cuenta que hay que introducir la cabecera del siguiente paquete antes que los bytes de dicha trama. El diagrama de bloques se muestra en la figura 4.14. Para ello se utiliza el core fifo generator [Cor12b] capaz de soportar interfaces de relojes diferenciadas para la lectura y escritura de la cola FIFO.

Finalmente, falta por conectar la salida del módulo concat a la interfaz del core PCIe y asegurar la integridad de los datos dados los dos dominios de reloj que se utilizan en la aplicación, 125MHz para el tratamiento de señales en el dominio de PCIe, 156.25MHz para el dominio de

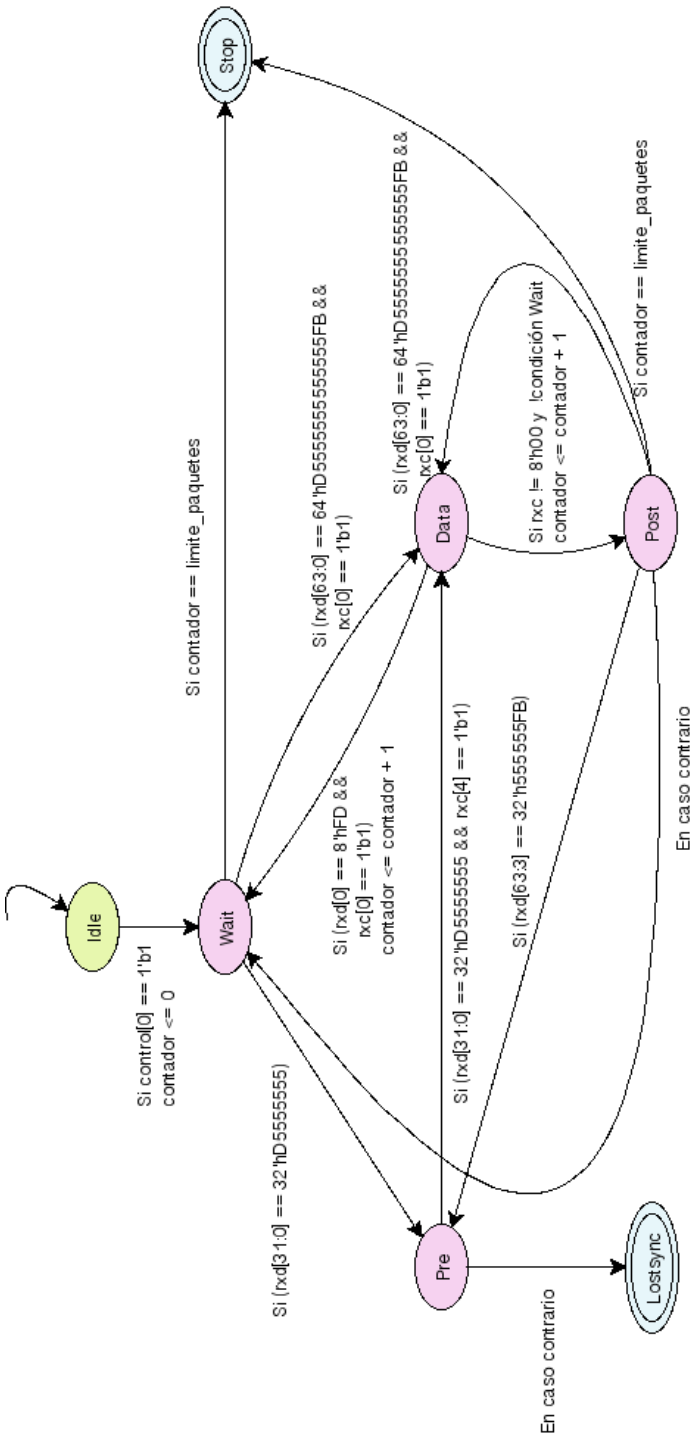


Figura 4.12: Máquina de estados para la interpretación del protocolo XGMII capturador de tráfico.

la interfaz a 10 Gbps.

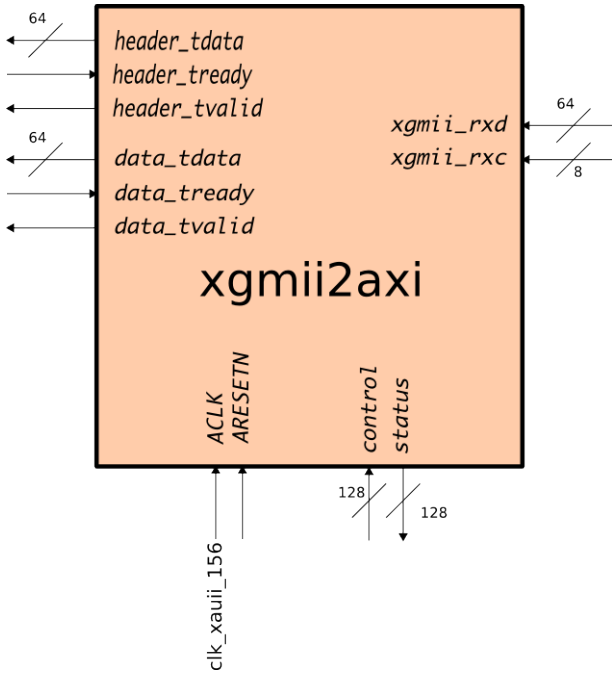


Figura 4.13: Diagrama de bloques del módulo xgmii2axi.

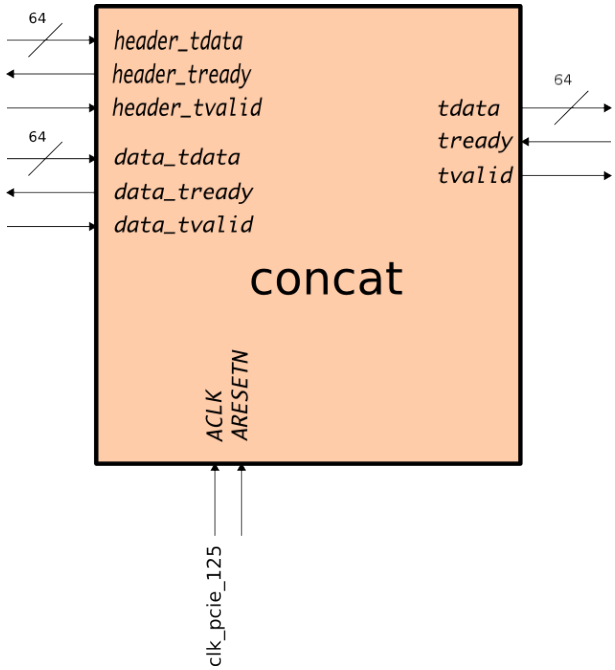


Figura 4.14: Diagrama de bloques del módulo concat.

4.4.2 Bus PCIe y comunicación con el anfitrión

En este apartado se presenta el core DMA de Northwest Logic [Log] de una manera más cercana al punto de vista de implementación que la mostrada en 4.3.2. La finalidad de este módulo consiste en la interpretación de los datos transferidos por el bus PCIe [PS05], ofreciendo a su salida un stream de datos en el formato AXI4-Stream (en el apéndice B se detallan los detalles del protocolo). En la figura 4.15 aparece un esquema de bloques interno del core. En la parte relativa al core PCIe se reutiliza el diseño inicial del proyecto NetFPGA.

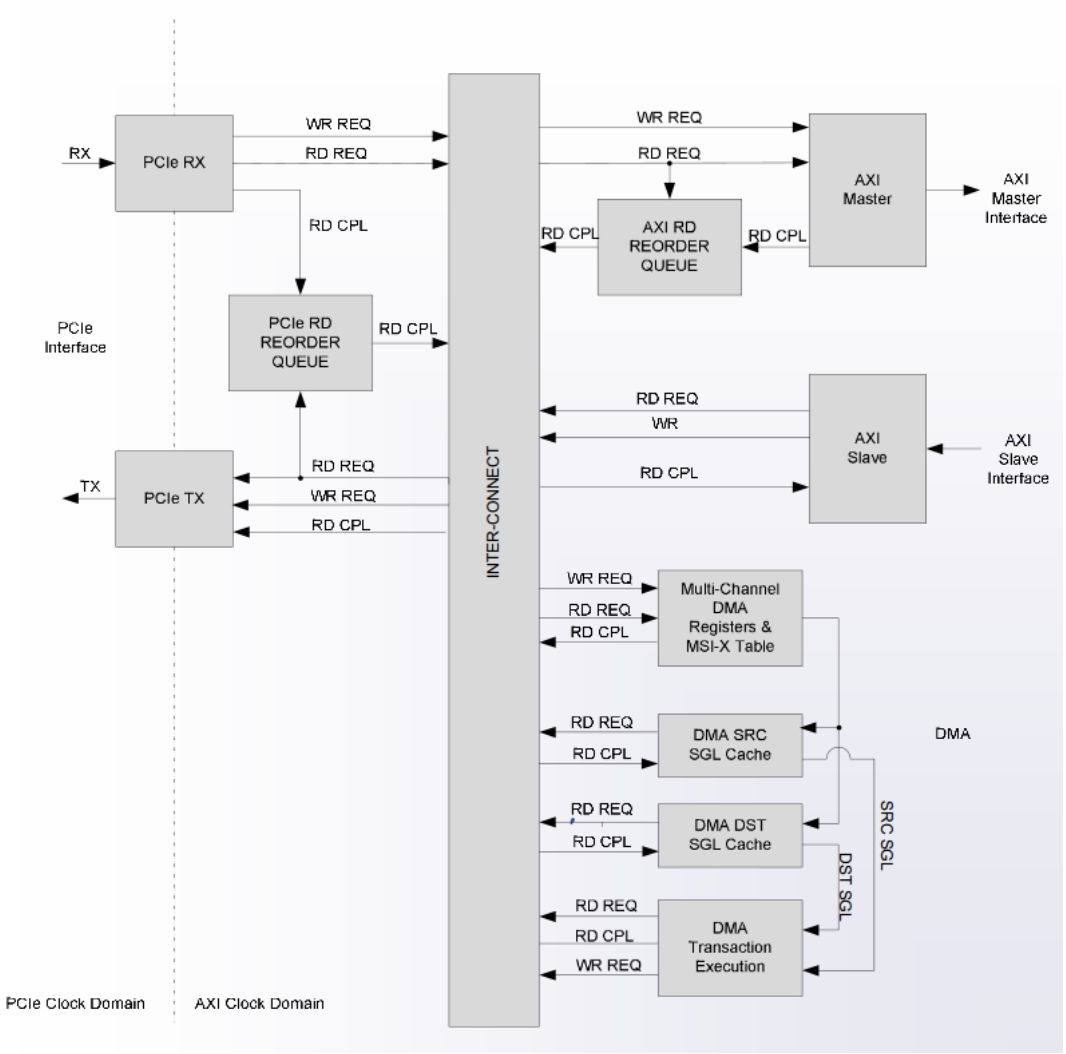


Figura 4.15: Diagrama de bloques del core DMA de Northwest [Log14].

Cabe destacar la presencia de dos interfaces AXI independientes, una maestra y otra esclava. Estas interfaces se corresponderán con las transferencias en dirección desde el anfitrión a la tarjeta y hacia el anfitrión desde la tarjeta respectivamente. A su vez, otros conceptos que no pueden ser modelados con las señales básicas del protocolo requieren de entradas adicionales. Ejemplo de estos nuevos conceptos que añade el core es la generación de interrupciones MSI/MSI-X cuando se termine de transferir un paquete (fragmento de datos delimitado por la activación de la señal LAST). Se distinguen, por tanto, las siguientes interfaces:

- Interfaz de administración. El propósito de la unidad de administración consiste en configurar y controlar el dispositivo, además de proveer información de estado y error. Ejem-

plos de parámetros configurables son, las ya mencionadas interrupciones y el tipo de éstas, payload máximo de un paquete PCIe y el tamaño de la petición máxima de lectura. Presenta un diseño propietario, por lo que mayor información del componente no puede ser expuesta. Las especificaciones concretas se encuentran en [Log14].

- Interfaz de DMA. Una interfaz por cada dirección, implementando un protocolo AXI4-Stream con las siguientes señales para el modo de lectura no direccionable: RVALID, RSTRB, RREADY, RDATA, RRESP (opcional en el estándar),RLAST (opcional en el estándar) y sus análogos para escritura WVALID, WSTRB, WREADY, WDATA, WRESP (opcional en el estándar) y WLAST (opcional en el estándar).

Se ha estudiado en el punto anterior como extraer los datos de un paquete ethernet en el protocolo XGMII. Para una primera aproximación que únicamente capture los datos en bruto, bastará con introducirlos en una cola FIFO. Como reloj de referencia en la escritura se usará el reloj de 156.25 MHz del protocolo XAUI. Como reloj de referencia en la lectura el reloj de 125 MHz, frecuencia empleada por el bus PCIe. Se tiene que la lógica de usuario deberá extraer los datos de la cola FIFO. Asentir la señal de VALID (indica contenido válido en la palabra transferida en el propio ciclo) y STRB (bytes válidos dentro de la palabra de ancho 128 bits) cuando se lea un nuevo dato en el bus AXI4-Stream destinado a la transferencia de datos hacia el host. Todo este proceso debe ser sincronizado con la señal READY de modo que no se comuniquen datos sin que el core haya sido capaz de procesarlos. La señal de LAST debe activarse una vez que se hayan leído $2^{19}B = 512KB$ de la FIFO para indicar que no se transmita más información en el descriptor actual.

4.4.3 Integración

A lo largo de la sección se han analizado individualmente distintos componentes. Se ha inspeccionado el core de DMA al igual que la conversión de los datos del protocolo XGMII a un formato propio (tamaño e interframe gap sucedidos de los datos de una trama). La interfaz de comunicación a todos estos dispositivos es AXI4-Stream. A su vez, queda por solucionar el problema a la comunicación de datos entre las distintas interfaces debido a los dominios de reloj. La solución aplicada queda contemplada en la figura 4.16.

Se generan tres colas fifo a través del core FIFO Generator de Xilinx [Cor12b]. La primera tiene por finalidad almacenar palabras de 64 bits que precederán a cada paquete. La segunda fifo, alberga los octetos asociados a la trama. Estas estructuras cuentan con un único dominio de reloj y son implementadas mediante block rams. Una tercera FIFO capaz de realizar la transferencia de datos entre los dos dominios es situada inmediatamente antes de la interfaz PCIe. En escritura se contempla el reloj utilizado por la interfaz de captura, un reloj de 156.25MHz, mientras que en lectura el reloj usado es el reloj de referencia para PCIe, a 125 MHz.

Se destaca la presencia de dos registros configurables mediante las señales de entrada app_mgt_addr, app_mgt_en, app_mgt_be y la señal app_mgt_data, que será de entrada para las señales que se encargan de escribir en el registro y de salida en el caso que se pretende leer desde registro. La funcionalidad de las señales es la siguiente:

- app_mgt_addr. Indica la dirección de memoria donde se quiere leer/escribir. En este caso, únicamente hay un registro para lectura y otro para escritura, ubicados en la dirección 0.
- app_mgt_en. Indica que en el dato referenciado por app_mgt_data hay algún octeto válido.
- app_mgt_be. Indica qué bytes en el dato referenciado por app_mgt_data son válidos.
- app_mgt_data. Palabra de 128 bits que se pretende leer/escribir.

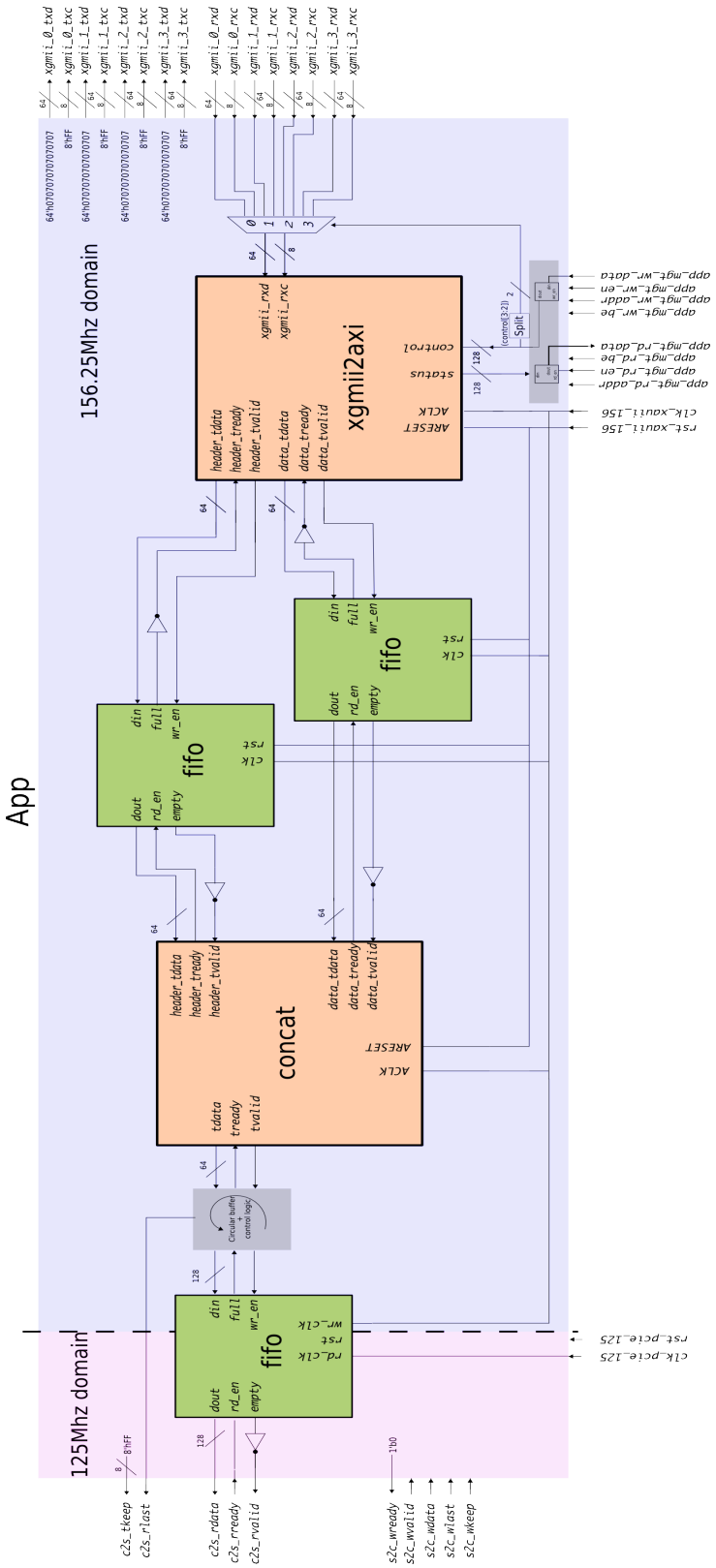


Figura 4.16: Diagrama de bloques del módulo app.

A través de esta interfaz se permite configurar detalles del comportamiento del modelo, al

igual que posibilita obtener datos en tiempo real (pérdida de paquetes o número de paquetes recibidos hasta el momento). Estos valores también se propagan a través del bus PCIe, utilizando una tercera interfaz AXI-Lite (similar al protocolo AXI4-Stream pero cuya principal finalidad es la transmisión de breves ráfagas de datos) del core PCIe para la lectura/escritura de palabras. En la figura 4.17 se contempla el diseño final una vez instanciados todos los elementos.

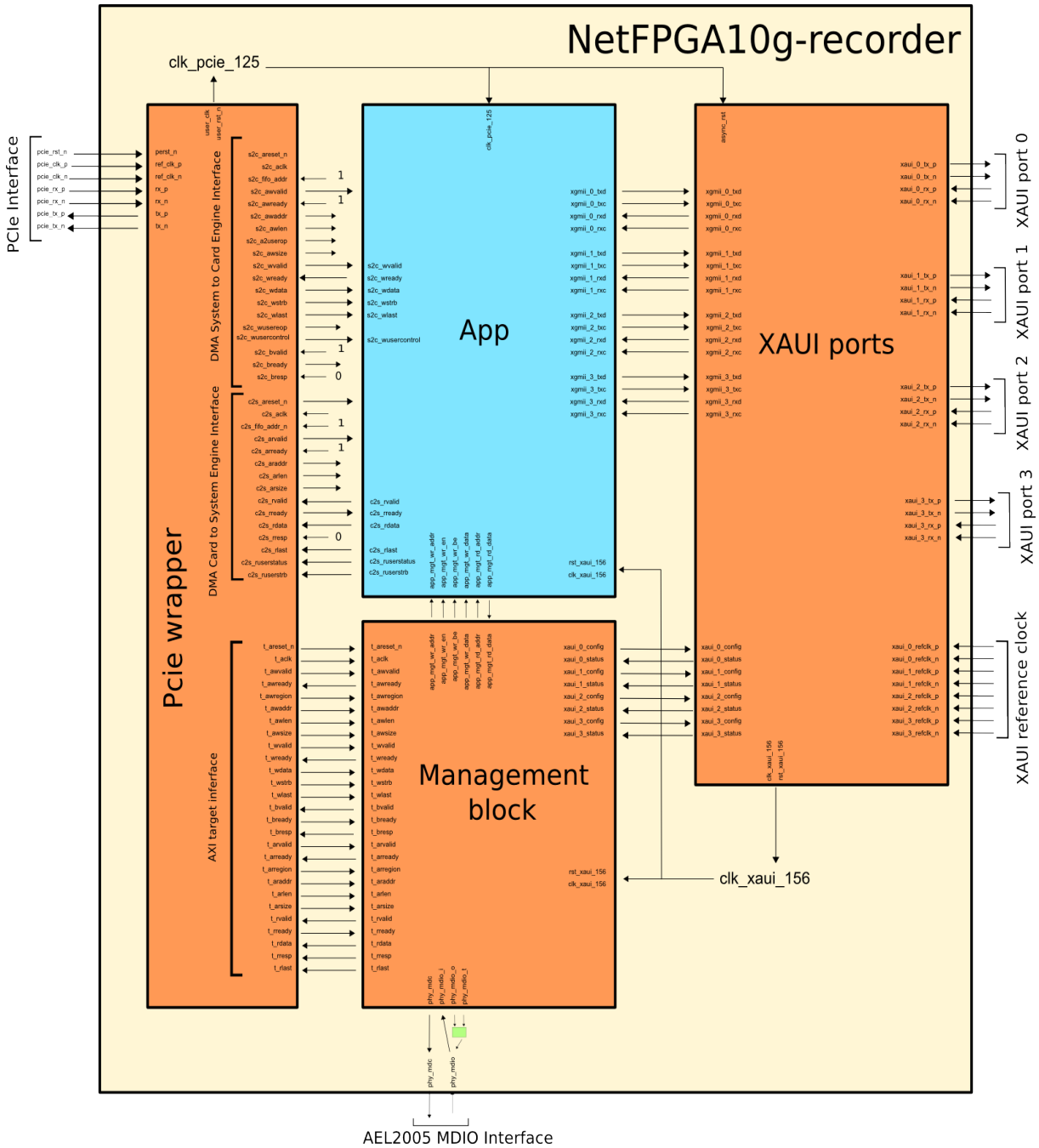


Figura 4.17: Arquitectura del diseño de captura de tráfico.

La tabla 4.3 muestra los detalles de uso de los recursos de la FPGA para el diseño de captura para su posible extrapolación a otros entornos.

Elemento lógico	Unidades utilizadas	Porcentaje respecto del total
Registros	28128	18%
LUTs (“Look up tables”)	24981	16%
Particiones (“slices”)	12412	33%
Block RAMs/FIFO	2070KB	20%
BUFG/BUFGCTRLs (“Globalbuffer”/“Global Clock Control Buffer”)	14	43%

Tabla 4.3: Recursos utilizados por el diseño hardware en captura.

4.4.4 Generador de tráfico 10 Gbps

El diseño presentado hasta el momento es completamente compatible para realizar la funcionalidad recíproca, es decir, la generación de tráfico a 10 Gbps. El abanico de posibilidades que ofrece una herramienta de este estilo es amplio, quizás mayor que el de un capturador. Se puede probar el comportamiento de un servidor frente a un tráfico masivo, simular un entorno malicioso e, incluso si se ha sido capaz de generar una traza en formato PCAP con unos tiempos de llegada siguiendo una función de distribución conocida, reproducir el tráfico bajo esa distribución.

El contenido del presente documento es referido a la captura en redes multigigabit ethernet, por lo que no se entrará en mayores detalles sobre las alternativas actuales, precios y características. Alguna de las utilidades presentadas en 2 llevan asociadas una parte de puesta en red de datos a la tasa indicada como PacketShader [SH11].

La arquitectura para el diseño sigue siendo la mostrada en la figura 4.2a: existe un diseño de usuario que lee de un fichero en el formato de la sección 4.2.1 los datos y los comunica mediante el driver propio hacia la tarjeta FPGA mediante el bus PCIe. En el nivel hardware se disecciona cabecera (tamaño e interframe gap) de los datos, transfiriendo estos últimos a través del protocolo XGMII. Se implementa una lógica capaz de añadir tantas palabras de IFG como sea necesario. Además se implementa una utilidad a nivel de usuario capaz de convertir un fichero PCAP al formato propio.

El equipo físico necesario no necesita modificarse para ejecutar esta nueva aplicación. El sistema de almacenamiento, uno de los principales cuellos de botella para este tipo de aplicaciones típicamente, será capaz de asegurar una mayor tasa de lectura que de escritura. Si en escritura no existen problemas, el experimento debería poder ser ejecutado sin problemas. La tasa del bus PCIe es similar en lectura y en escritura (la tasa teórica es la misma pero existe la salvedad de que el protocolo basado en el intercambio de mensajes no es completamente simétrico). Las modificaciones deben realizarse en el diseño hardware y, concretamente, dentro del módulo “app”. Se plantea una arquitectura como la mostrada en la imagen 4.18.

El módulo “app” en esta ocasión es más reducido. Se genera un único core FIFO, encargado de realizar la transición entre los dos dominios de reloj. El módulo axi2xgmii es consciente del formato de los datos, por lo que esperará en un estado inicial una palabra indicando tamaño y número de bytes de inactividad tras enviar los datos. Cuando detecta esta secuencia inicial escribe en la interfaz XGMII el preámbulo 64’hD5555555555555FB (en la señal de datos) mientras que los bits de control asociados se establecen 8’h01. A continuación los bits de control de establecen a 0 mientras se comunican el resto de valores de la trama. Se termina la operación concatenando un byte FB y tantos 07 (idle) como número se haya especificado en el interframe gap. En este último tramo los bits de control asociados están a 1. Para más detalles consultar la tabla 4.1. Esta aproximación sufre pequeñas variaciones si la primera palabra a transferir

no está en el primer byte de la palabra de 64 bits. La única posibilidad alternativa es que se encuentre en el 4º byte. Es conveniente recordar que XGMII transmite tanto en flanco ascendente como descendente un total de 32 bits pero el inicio de un paquete debe estar fijado a la línea 0. El core XGMII usado engloba esos datos para ofertarlos en un flanco ascendente. En tal caso habría que considerar que hay que "desplazar" el inicio del paquete. La primera secuencia sería 64'h555555FB07070707, en el siguiente ciclo se transmitiría 64'hD3D2D1D0D5555555, donde D3D2D1D0 se corresponden con los primeros 32 bits del paquete, y así sucesivamente. El caracter de terminación nuevamente es 8'hFD y puede estar situado en cualquier línea.

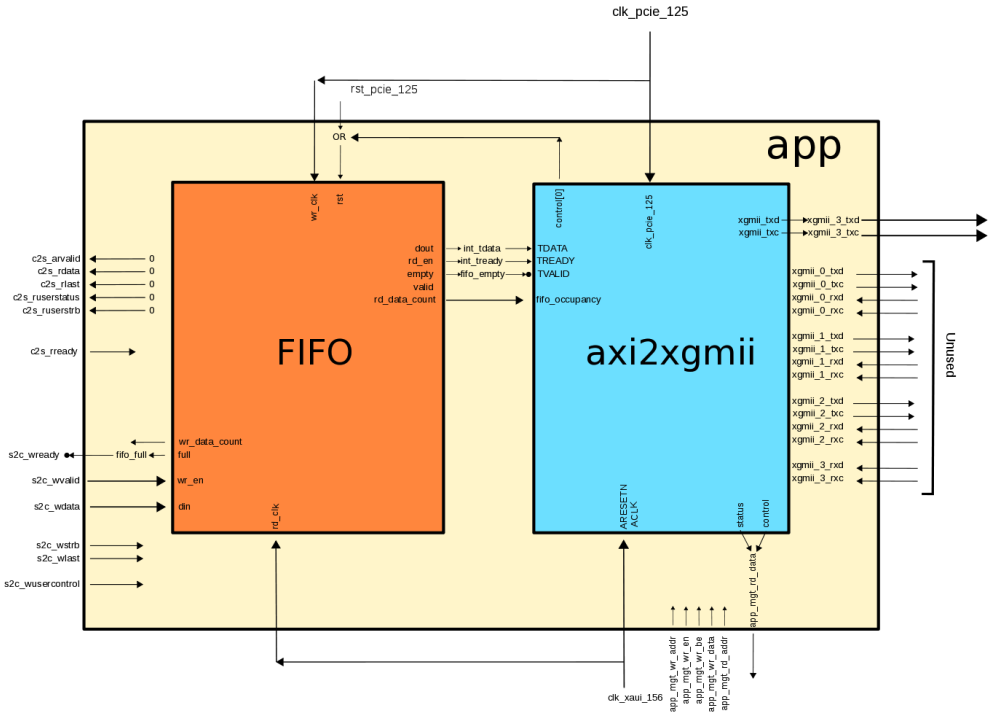


Figura 4.18: Diagrama de bloques del módulo app para la reproducción a 10 Gbps.

Por último, en la tabla 4.4 se informa de los recursos de la FPGA utilizados en el sistema de generación de tráfico, en contrapartida a los necesitados por el diseño de captura (tabla 4.3).

Elemento lógico	Unidades utilizadas	Porcentaje respecto del total
Registros	27661	18 %
LUTs ("Look up tables")	26593	17 %
Particiones ("slices")	13286	35 %
Block RAMs/FIFO	5652 KB	48 %
BUFG/BUFGCTRLs	14	43 %

Tabla 4.4: Recursos utilizados por el diseño hardware de generación de tráfico.

4.5 Diseños independientes a nivel de usuario

En esta sección se presentan programas en el nivel de usuario relativamente independientes pero que guardan una especial ligadura con el capturador de tráfico ya sea porque modifican

el comportamiento del diseño en la FPGA o por su utilidad a la hora de elaborar un sistema de pruebas.

Los diferentes programas elaborados son:

- Programas de control/estado.
- Configuración del chip de medio físico.
- Conversor de PCAP a formato simple.
- Generador de tráfico.
- Reproductor de tráfico aleatorio. Incremento de flujos.

4.5.1 Programas de control/estado

En el catálogo inicial de requisitos (sección 3.2) se expresa la necesidad de implementar distintos estados para el diseño hardware. A través del protocolo (AXI4-Lite) la FPGA es capaz de proveer información al programa de usuario mediante la lectura de registros en unos desplazamientos prefijados del BAR1. Los datos de estado se encapsulan en un único registro de 128 bits mostrado en la imagen 4.19.

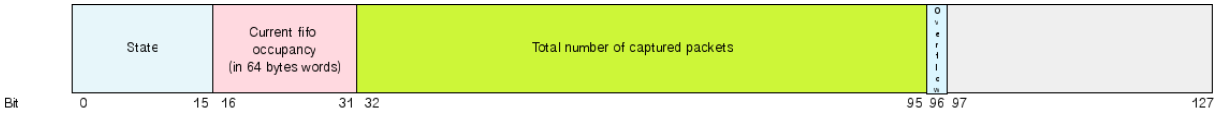


Figura 4.19: Palabra de estado del capturador.

La información se corresponde con:

- Bits 15 a 0. Identifican el estado actual de la máquina de estados, pudiendo corresponderse con:
  - RESET = 16'b0000000000000001, el usuario está comunicando un reinicio de las máquinas de estado mediante la palabra de control y no ha finalizado su petición.
  - IDLE = 16'b0000000000000010, el diseño está listo para comenzar a capturar pero el usuario aún no ha activado el bit de comienzo en la palabra de control.
  - WAIT = 16'b0000000000000100, el diseño está esperando datos de la red. Actualmente no se está transmitiendo ninguna información válida.
  - PRE = 16'b0000000000001000, se ha detectado una secuencia de inicio de paquete del protocolo XGMII: 32'h555555FB. A la espera del final de la secuencia, 32'hD5555555.
  - POST = 16'b0000000000010000, se ha localizado el final de un paquete pero se deben volcar los bytes de la última palabra válida.
  - DATA = 16'b0000000001000000, se están recibiendo datos válidos de la red.
  - STOP = 16'b0000000010000000, se ha terminado de procesar el número total de paquetes especificados al módulo mediante la palabra de control.
  - LOSTSYNC = 16'b0000001000000000, ha habido algún problema de sincronización y el diseño hardware ha fallado.
  - UNDERRUN = 16'b0000010000000000, el diseño está perdiendo paquetes.

- Bits 31 a 16. Representa la ocupación de la FIFO de datos usada para asegurar la integridad de los datos entre las dos frecuencias empleadas en palabras de 64 bits. Un valor cercano al máximo (4096) puede indicar posibles problemas de desbordamiento en un futuro.
- Bits 95 a 32. Entero de 64 bits con el número total de paquetes detectados por el módulo.
- Bit 97. Flag de estado indicando si la FIFO se ha llenado en algún momento y se han descartado datos.

A su vez, operaciones básicas que comuniquen al diseño que se desea comenzar con el inicio de la captura, especificar el número total de paquetes a capturar, el nivel mínimo de ocupación de la FIFO de datos antes de transmitir datos o el reinicio de la máquina de estados y las unidades lógicas usadas. El registro de control cuenta con un total de 128 bits con los siguientes campos (figura 4.20).

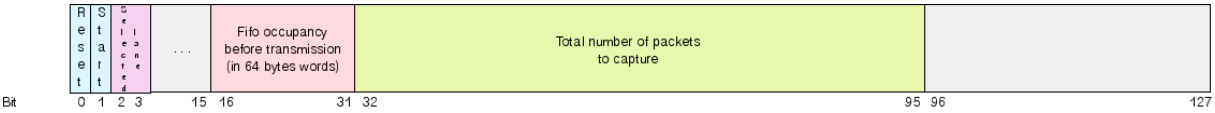


Figura 4.20: Palabra de control del capturador.

- Bit 0. Flag de control “restart”: indica un reinicio de las máquinas de estados y FIFOs.
- Bit 1. Flag de control “start”: indica al diseño que debe comenzar con el proceso de captura (requiere que el número de paquetes por capturar contenga un dato válido).
- Bit 3 a 2. Se utilizan para seleccionar la interfaz a 10 Gbps de la tarjeta FPGA a utilizar. Su configuración debe realizarse antes del comienzo de la captura y el comportamiento no esta especificado si se modifica en ejecución.
- Bit 31 a 16. Ocupación de la FIFO antes de empezar a transmitir datos hacia el anfitrión.
- Bit 95 a 32. Entero de 64 bits con el número total de paquetes a capturar por el módulo.

Se elabora, por tanto, unos programas que encapsulen toda la escritura/lectura en registro mediante la invocación con los parámetros deseados para cada uno de los campos sin necesidad de que el usuario deba conocer el formato de las palabras.

4.5.2 Configuración del chip de medio físico (PHYceiver)

Una instancia de la capa física en el modelo OSI conecta un dispositivo de la capa de enlace, con frecuencia llamada MAC, con un medio físico como pudiera ser la fibra óptica o el cable de cobre. Un dispositivo PHY incluye típicamente una subcapa de codificación física y una dependiente de la capa medio de físico. Un chip de medio físico (PHYceiver) se encuentra comúnmente en los dispositivos ethernet, que implementa el envío y la recepción de tramas (ethernet) a nivel hardware. Se utiliza en conjunto con una interfaz independiente del medio (MII) que permita aislar al diseño del medio físico empleado.

Se diseña una herramienta capaz de seleccionar como posibles medios de transmisión fibra óptica de corto alcance (10GBASE-SR [Soc12]) y cableado twin-axial (10GBASE-CR) en un SFP+ transceptor que puede ser conectado a cualquiera de las 4 interfaces que acompañan a la tarjeta NetFPGA. La configuración debe ejecutarse únicamente tras la conexión física de la interfaz para que el sistema sea capaz de detectar el enlace.

4.5.3 Programas asociados a la generación de tráfico

El programa encargado de leer de un fichero en el formato de la sección 4.2.1, formato simple, nuevamente implementa un modelo productor-consumidor. Mientras que en captura el productor era el diseño hardware y el consumidor el diseño de usuario, en esta ocasión los papeles se invierten.

El programa de usuario es sencillo. Lee de disco y comunica la cantidad de bytes leídos y la dirección al driver propio. Esta lectura de disco es volcada a la región de memoria proporcionada por las huge pages, compartidas entre el diseño de usuario y driver. En esta ocasión únicamente existe un hilo (dado que la operación de envío de datos al driver es no bloqueante) pero hay que tener en consideración no estar escribiendo sobre una región de memoria que pudiera no haber sido enviada aún (caso en que el diseño software trabaja más rápido que diseño hardware). El algoritmo queda reflejado en el cuadro 4.11.

Para la implementación es necesario la creación de una nueva nueva rutina IOCTL que indique el último descriptor que ha sido enviado por el driver. Esta tarea es sencilla ya que el driver internamente necesita tener constancia del puntero desde el primer descriptor sobre el que se desconoce si ha terminado hasta el último que el usuario ha pedido enviar. Basta con devolver el índice del primero menos 1 módulo número total de descriptores.

Como posibilidad adicional se implementa la reproducción en bucle, es decir, generar el tráfico referenciado por un fichero un número especificado de iteraciones. Esta opción puede ser útil si el fichero es suficientemente descriptivo para la prueba que se persigue.

```
1  UnidadDeLectura  ← MÚLTIPLO DESCRIPTOR
2  ÚltimoDescriptorComprobado  ← NúmeroTotalDescriptores - 1
3
4
5  Mientras no fin de fichero
6      Leer UnidadDeLectura datos de fichero
7
8      Comunicar driver tamaño y posición
9
10     Mientras (posición + UnidadDeLectura)/(TAMAÑO DESCRIPTOR)
11             ≡ ÚltimoDescriptorComprobado módulo (TAMAÑO DESCRIPTOR)
12         ÚltimoDescriptorComprobado  ← Preguntar driver último
               descriptor enviado.
13     fin mientras
14 fin mientras
```

Cuadro 4.11: Algoritmo programa de usuario para la reproducción de datos.

Sin embargo, si no importa tanto el contenido como el tamaño del paquete, se crea otro programa adicional, sendRandom, que dado un tamaño y un número de flujos distintos, genera tantos paquetes como se hayan indicado (hasta un máximo limitado por la memoria asociada a las huge pages). A nivel de usuario no es posible generar paquetes de manera dinámica en el equipo presentado. El cálculo del CRC a nivel de enlace, es una operación pesada que prefiere hacerse en hardware. Como en este caso no se ha provisto al diseño de tal funcionalidad, su procesamiento en software evita que se consiga la tasa esperada. Para solventar este problema la alternativa propuesta es la generación de paquetes aleatorios (en el sentido del payload, cabeceras ethernet, IP y TCP tienen valores ficticios pero coherentes) sobre la región apuntada por las huge pages. Tras un primer procesamiento de creación de los paquetes, el mismo programa comunica al driver la necesidad de enviar esa región de memoria indefinidamente (o hasta alcanzar un umbral especificado por el usuario). Esta técnica puede ser útil para la

medición de prestaciones de otras herramientas. En este caso particular, la reproducción de paquetes de tamaño mínimo y máximo facilitará la medida de prestaciones del capturador de tráfico.

4.6 Resumen

Se ha abordado la descripción del diseño del capturador de una manera descendente. Se proporcionan las especificaciones del equipo físico sobre el que se instala el prototipo, de aproximadamente unos 3000€. Se especifica como sistema operativo de referencia CentOS 6.5 y se mencionan las herramientas necesarias para la creación del entorno. Se detalla como instalar el sistema de huge pages, las herramientas necesarias para la instalación con la plataforma NetFPGA [Uni], Xilinx Lab Tools, y se sugiere el aislamiento de ciertas CPUs, así como la desactivación de las opciones de hyperthreading.

Se describe el diseño a nivel de usuario, que escribe en disco los octetos provenientes de la red, al igual que el driver encargado de realizar de intermediario entre la FPGA y el código de usuario. Del diseño hardware se detallan aspectos pertenecientes a la capa de nivel 1 del protocolo OSI, las máquinas de estado que están involucradas en la recepción, métodos usados para solventar la transferencia de datos entre distintos dominios de reloj y los módulos desarrollados. Entre estos últimos cabe destacar el core PCIe [Uni] y el core DMA [Log] que son herramientas desarrolladas por terceros. Basándose en esta arquitectura, se brinda información sobre un generador de tráfico a 10 Gbps que servirá como referencia para medir las prestaciones en captura.

5 Rendimiento

En este apartado se valora de manera objetiva los resultados obtenidos. Para ello en primera instancia se obtienen unas cotas aproximadas para los valores esperables del diseño. Posteriormente, se muestran las medidas reales del sistema y se busca una justificación a los resultados.

5.1 Estimaciones teóricas

En primer lugar debemos distinguir entre los distintos elementos involucrados en todo el proceso de captura cuáles suponen realmente un posible cuello de botella. Por ejemplo, las interfaces de red aseguran un tasa máxima de 10 Gbps. Es decir, no es viable obtener ningún rendimiento superior a este valor. No depende del diseño del programador mejorar su rendimiento dado que es una limitación de fábrica. En cualquier caso, nos gustaría que no se experimentase ninguna penalización a esa velocidad a lo largo de todo el proceso. Cada uno de los elementos involucrados en el sistema deben ser capaces trabajar a una tasa mayor o igual a la ya citada. Realicemos un análisis de los elementos de la cadena:

(a) Diseño en verilog. El módulo que se ejecuta en la FPGA tiene como objetivo convertir la señal del protocolo XAUI a AXI4-Stream. Posteriormente, la información se almacena en la tarjeta FPGA para su posterior transferencia vía operación DMA.

En este recorrido la única posible limitación se trata de la memoria utilizada para el guardado de los datos dado que la conversión puede realizarse en el mismo ciclo de reloj. Se simula en primera instancia el caso en el que se emplean los tres módulos QDR disponibles. Cada uno de los mismos cuenta con un ancho de bus de 36 bits y operan a una frecuencia de 300MHz (los datos técnicos de cada módulo individualmente quedan reflejados en la tabla 5.1).

Característica	**Valor**
Frecuencia	300MHz
Frecuencia máxima	333MHz
Densidad	18-144MB
Anchura del bus	36b
Longitud de ráfagas	4
Latencia de lectura	1.5 ciclos
Voltaje del núcleo	1.8V

Tabla 5.1: Especificaciones memoria QDR.

Es decir, a la frecuencia de 300MHz se ofrece una tasa teórica al disponer de tres memorias (suponiendo inicialmente un bus de 1 bit) de

$$300MHz * 3 = 900Mb/s$$

Como el ancho de banda es de 36 bits, el rendimiento total se sitúa en:

$$900Mb/s * 36 = 30{,}845Gbps$$

Para una ráfaga de 4 palabras el tamaño total de bits a ser transferido es de $36*4 = 144bits$. Aplicando que en cada ciclo se transmiten 4 palabras y existen 3 módulos de memoria:

$$36(\text{anchura del bus}) * 4(\text{memoria QDR}) * 3(\text{número de memorias}) = 432bits/ciclo$$

Es decir, para leer $432bits$, contando con una latencia de ciclo y medio para comenzar la operación, el total de tiempo operativo es de 2 ciclos y medio, un 1/2,5 = 40 % del total. Añadiendo un 5 % de overhead extra para acciones de refresco, la eficiencia total es cercana al 35 %, unos $10{,}8Gbps$.

La anterior tasa es la máxima que se puede alcanzar en lectura intercalada con escrituras. Para un capturador a 10 Gbps se trata de una limitación considerable puesto que la incorporación de escrituras en memoria provoca una reducción del rendimiento. Posibles soluciones al problema sería utilizar memorias que contasen con un ancho de bus mayor, incrementar la longitud de las ráfagas o la frecuencia de las mismas. Como la plataforma hardware no ofrece soporte para mejorar estos atributos nace la idea de almacenar la información en una memoria local, sin tener que recurrir a módulos externos.

Una memoria RAM síncrona no puede leer/modificar/escribir en un único ciclo de reloj. Las FPGAs de Xilinx cuentan con memoria RAM por bloques (del inglés "block RAMs") que pueden aplicar pipeline sobre las operaciones de escritura para obtener un rendimiento de una lectura/escritura/modificación por ciclo. Para implementar tal finalidad la memoria cuenta con dos puertos, A y B. En la figura 5.1 aparece el esquemático del componente.

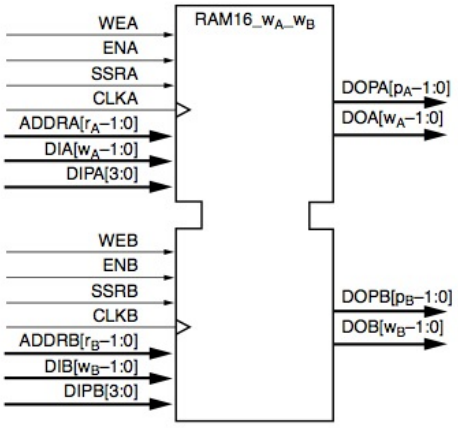


Figura 5.1: Esquemático block RAM de doble puerto. Fuente XAPP463 [Xil05].

Cada uno de los puertos posee señales de control diferenciadas (existe la posibilidad de trabajar a distintas frecuencias de reloj). A continuación se realiza una breve descripción de cada una de las señales:

- Señal de activación de escritura en el puerto: WE (write enable). Indica si el puerto escribirá el dato DI en la dirección apuntada por ADDR.
- Dirección de lectura/escritura: ADDR.
- Reloj del puerto: CLK.
- Señal de reset síncrona: SSR.
- Señal de entrada: DI (data input). En caso de escritura el dato a albergar.
- Paridad de la señal de entrada: DIP (data input parity).
- Señal de salida: DO (data output). En caso de lectura devuelve el dato solicitado.
- Paridad de la señal de salida: DOP (data output parity).

A pesar de ofrecer una menor capacidad de almacenamiento, el uso de utilizar la memoria local (en forma de block RAMs), asegura que no haya penalizaciones en la tasa exigida para el proyecto dado que se permite tanto leer como escribir datos en el mismo ciclo de reloj. Las especificaciones para una memoria block RAM con interfaz AXI4-Stream aparecen descritas en la tabla 5.2 [Cor12a].

Característica	**Valor**
Frecuencia máxima	300MHz
Densidad	De $2 \cdot 10^6$ a $9 \cdot 10^6$ palabras (limitado por los recursos de la placa y anchura del bus)
Anchura del bus	Entre 1 y 4608 bits
Latencia en acceso	"Zero delay datapath", tras una petición de lectura/escritura en el ciclo siguiente se puede completar la operación
Canales	Caminos independientes para lecturas y escrituras

Tabla 5.2: Especificaciones memoria block RAM.

Bajo estas características, tomando como anchura del bus 64 bits a una frecuencia de 156.25MHz (reloj de referencia de las interfaces de red), se tiene una tasa de:

$$156{,}25MHz * 64 = 10000Mb/s = 10Gb/s$$

El tamaño mínimo de bus necesario para la aplicación se fija por tanto en 64 bits, que coincide con las dimensiones presentadas en el apartado 4.4. Análogamente se comprueba como para la frecuencia de 125MHz, una anchura del bus de 128 bits para asegura la tasa de 10 Gbps,

$$125MHz * 128 = 16000Mb/s = 16Gb/s$$

que se trata de la máxima velocidad de transferencia que pueden asegurar 8 líneas PCIe 1.0.

(b) Transferencia DMA. Al disponer de una tarjeta de 8 líneas en PCIe 1.0, el máximo ancho de banda unidireccional (hacia la tarjeta/desde la tarjeta) por línea es de 2, 5 Gbps [PS05]. En la dirección de la tarjeta hacia el anfitrión la máxima transferencia teórica es

$$2{,}5Gb/(s * lines) * 8\ lines = 20Gbps$$

Sin embargo, esta estimación debe ser afinada dado que el protocolo PCIe está basado en paquetes y utiliza una codificación $8b/10b$. Fijemos los parámetros de referencia de la tabla 5.3.

Característica	**Valor**
Payload máximo	128 bytes
Petición de lectura máxima	128 bytes
Datos apuntados por el descriptor	4KB
Un ACK por cada TLP-DLLP	8B de overhead

Tabla 5.3: Datos de referencia PCIe.

Para estos valores disponemos de las penalizaciones mostradas en la tabla 5.4.

Overhead en la transacción	Overhead de ACK	Aclaración
Lectura de memoria: obtención del descriptor $\frac{20}{4096} = \frac{\mathbf{0,625}}{\mathbf{128}}$	$\frac{8}{4096} = \frac{0,25}{128}$	Un descriptor apunta a una región de 4KB de datos. Además hay 20B extra debido a las cabeceras TLP y 8 bytes de Data Link Layer Packets (DLLP).
Comunicación de información relativa a los datos: Información relativa al descriptor $\frac{20+32}{4096} = \frac{1,625}{128}$	$\frac{8}{4096} = \frac{\mathbf{0,25}}{\mathbf{128}}$	La cabecera para transmitir el descriptor es de 20 bytes; el tamaño del descriptor de 32 bytes (por definición del core. No confundir con tamaño de la región apuntada por el descriptor: 4KB).
Escritura en memoria: motor DMA $\frac{\mathbf{20}}{\mathbf{128}}$	$\frac{8}{128}$	El tamaño máximo para el payload es de 128 bytes. Se considera la configuración del motor de DMA.
Escritura en memoria: actualización descriptor $\frac{20+12}{4096} = \frac{1}{128}$	$\frac{8}{4096} = \frac{\mathbf{0,25}}{\mathbf{128}}$	Consideramos la cabecera de 20 bytes del paquete y la actualización del descriptor (12 bytes).

Tabla 5.4: Penalización en PCIe.

Resumiendo, por cada 128 bytes de datos enviados desde la tarjeta al sistema hay un overhead total de 21,875 bytes (indicados en negrita siguiendo el patrón: en lecturas de memoria se contabiliza el overhead de la cabecera, en comunicación de información hacia el anfitrión y escrituras en memoria se considera el overhead de ACK).

PCIe utiliza una codificación 8b/10b. Por tanto, el rendimiento del protocolo para 8 líneas es:

$$20{,}0 Gbps \frac{8}{10} = 16\ Gbps$$

Velocidad a la que debe penalizarse el overhead. En porcentaje supone un total del

$$\frac{9,875}{128 + 9,875} \approx 7,16\,\%$$

del tiempo. Se tiene, por tanto, la estimación teórica de

$$16\ Gbps * (1 - 0,07162) = 14,854\ Gbps$$

para la tasa de transferencia en PCIe. Este componente puede actuar de cuello de botella para la captura en redes a 10 Gbps. Además, para redes multigigabit a mayores frecuencias impone una limitación que deberá ser solventada mediante el uso de la segunda/tercera generación de PCIe [PS10]. En esta nueva versión cambian aspectos como

la codificación aplicada (la codificación 8b/10b evoluciona y únicamente emplea dos bits de redundancia por cada 128, 128b/130b en la tercera generación) y se aumenta la transferencia máxima teórica por cada línea (asciende de los 250MB/s a 500MB/s y 1GB/s respectivamente).

(c) Driver y diseño de usuario. Se esperaría que el "overhead" introducido por el driver y el diseño del usuario no fuera mayor que el incorporado por el sistema operativo al desencapsular los paquetes TLP. Incorporan un pequeño retraso que a fines prácticos supondremos que son despreciables en el rendimiento global. La ejecución del programa en un sistema multicore donde se puede establecer la afinidad de los distintos hilos a distintas unidades de proceso favorece que no haya penalizaciones por la obtención de recursos y que siempre que un proceso no esté bloqueado por motivos de entrada/salida pueda estar operando. La opción de arranque "isolcpus" permite que el planificador de linux ignore una lista de procesadores a la hora de realizar la asignación de tareas y una correcta configuración puede ser motivo de una mejora significativa en las medidas obtenidas.

(d) Escritura en disco. En este apartado pueden aparecer problemas al igual que en PCIe. Comprobaremos qué requisitos teóricos son necesarios para desempeñar la actividad. Se dispone de 8 discos SSD Samsung SSD 840 EVO cuyas especificaciones quedan recogidas en la tabla 5.5 [Sam13]. Para obtener el máximo rendimiento de los componentes se construye un RAID de nivel 0. Un RAID-0 no incorpora redundancia en los datos almacenados y es probable que ante el fallo de alguno de los miembros el sistema no se recupere de la pérdida de información. Para las pruebas de rendimiento esta es una opción viable dado que asegura el mayor rendimiento en escritura comparado con el resto de tipos de RAIDs (típicamente la carga de escritura se podrá dividir igualitariamente entre los distintos volúmenes, ver figura 5.2). La mejora de rendimiento consiste en realizar un particionamiento de los ficheros, en fragmentos de tamaño estático denominados como "stripe" y configurable durante la creación del RAID (generalmente es un valor comprendido entre 16KB y 256KB aunque puede estar sometido a cambio).

Síntesis de las especificaciones
Conexión SATA 6Gb/s.
512 MB memoria DDR2 SDRAM.
Memoria NAND flash a 400Mbps.
Capacidad 250 GB.
Rendimiento en operaciones de lectura/escritura: • Lectura secuencial: máximo 540 MB/s. • Escritura secuencial: máximo 520 MB/s.
Soporte para TRIM.

Tabla 5.5: Resumen especificaciones de los discos SSD Samsung SSD 840 EVO.

La estimación del rendimiento en escritura para n discos sería cercana a n veces la tasa de un disco unitario (usando los 8 discos disponibles debería alcanzarse un máximo de $520MB/s * 8 = 4160MB/s$) dado que $n-1$ unidades más de datos pueden ser escritas en el mismo tiempo. Bien es cierto, que existe cierta penalización de los controladores para realizar la división de los ficheros en las transacciones. Su comportamiento real se estudia en 5.2.2. Una observación importante que revelan las pruebas empíricas es que la penalización debida a un tamaño inadecuado de stripe puede ser realmente influyente

en el rendimiento. Si se escoge un tamaño excesivamente grande las ventajas de utilizar un RAID se pueden perder. Si por el contrario el tamaño especificado es ínfimo, desembocará en un excesivo overhead a la hora de realizar el particionamiento de los datos.

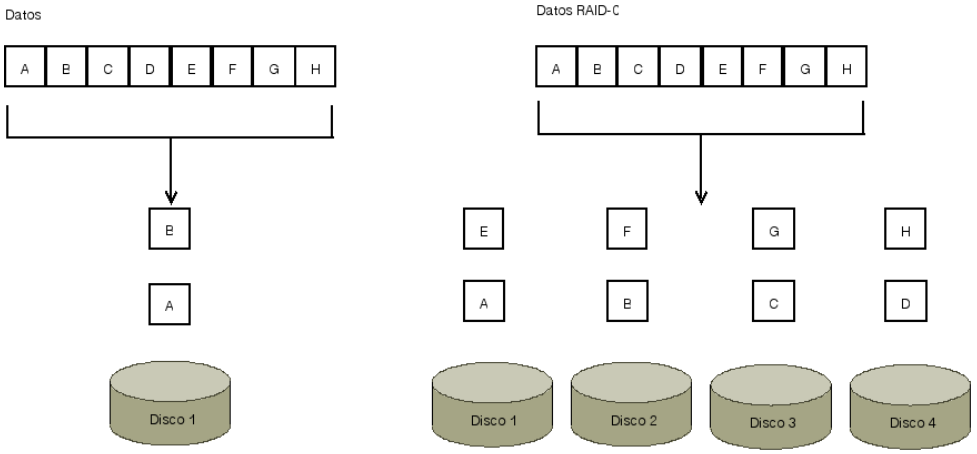


Figura 5.2: Comparación escritura en disco y RAID0 de 4 discos.

Una vez seleccionado el modelo de RAID que se aplicará en el diseño cabe la opción de realizar su montaje mediante un controlador a nivel hardware o a nivel software. Cada uno de ellos ofrece sus respectivas ventajas. Hay argumentos para inclinarse por cada una de las opciones. El principal argumento que se suele utilizar para la defensa de emplear un controlador software es el precio. No se necesita ningún componente adicional. Sin embargo, el no conseguir la tasa causaría tener que añadir nuevos discos al array por lo que no queda claro que realmente suponga un ahorro en el presupuesto.

Para los experimentos se dispone de una controladora Adaptec RAID 6805 que dispone de tecnología PCIe de segunda generación y un total de 8 puertos SATA a 6Gb/s. La implementación del RAID-0 que se realiza a nivel hardware esconde totalmente al sistema operativo el modelo de almacenamiento subyacente.

A nivel teórico si es necesario realizar unas observaciones previas:

- Los controladores RAID tienen una cantidad de memoria muy limitada (en este caso, 512MB), utilizados como caché para lecturas y/o, escrituras. En servidores equipados con SSDs no es extraño observar que el rendimiento es inferior con la caché habilitada porque la lógica no es igual de rápida que los discos. Existen controladoras RAIDs específicas para discos de estado sólido con las que no se ha podido contar para realizar las mediciones. Estos elementos persiguen alcanzar el máximo rendimiento posible en los arrays de discos de estado sólido y aplican técnicas de optimización pensadas especialmente para este tipo de unidades de almacenamiento.

5.2 Resultados empíricos

En este apartado se validan las estimaciones realizadas frente al comportamiento real del programa, informando de las posibles deficiencias encontradas y su justificación. Se analiza la tasa de transferencia de datos a través del bus PCIe y la tasa de escritura del RAID creado. Para atender al correcto funcionamiento del proyecto bastará con simular el entorno en un escenario real, atendiendo a los posibles registros de estado del diseño creado para valorar si realmente se cumple con la tasa exigida.

5.2.1 Tasa de transferencia a través del protocolo PCIe

Para este escenario no se depende estrictamente del diseño hardware del capturador. Se crea un diseño de referencia cuya única finalidad sea la de poner a prueba la tasa de transferencia ofrecida por los siguientes componentes:

- Core DMA de Northwest Logic [Log] y core PCIe de NetFPGA [Uni].
- Driver propio desarrollado. Se elabora una versión particular para medir el rendimiento,
  - Una primera versión capaz de configurar una transferencia de 8GB de datos sin intervención del diseño de usuario. En esta ocasión se intenta medir exclusivamente la tasa conseguida excluyendo la interacción con el driver y cuestiones propias del sistema operativo (planificación de procesos, invocación de la rutina encargada de atender las interrupciones, etc.) que intervienen en la sincronización con el diseño de usuario.
  - La versión completa del módulo, de modo que se puedan observar porciones de código no optimizadas o poco eficientes.
- Diseño de usuario. Se modifica el comportamiento para que en lugar de volcar sobre un medio físico, los datos se escriban en memoria principal. Es decir, se monta como un sistema de ficheros la memoria principal del ordenador (atender al tipo tmpfs del comando mount) y se pide escribir sobre esta región. Para tener datos comparables con la versión que no depende del diseño de usuario (módulo capaz de realizar la transferencia), se transmiten también un total de 8GB desde la placa hacia el anfitrión.

El diseño creado para la FPGA es sencillo, por cada ciclo de reloj se incrementa un contador, que es transferido a través del protocolo AXI4-Stream hacia la interfaz esclava del core (interfaz C2S). De este modo cada ciclo de reloj (siempre que sea posible porque el core este aceptando datos, READY activo), se habilita la señal de VALID y se transmiten 128 bits al core. La señal de LAST se activa cada 512KB. Los datos obtenidos quedan contemplados en la tabla 5.6.

Tasa de transferencia para el diseño implementado a nivel de driver	$14{,}12 Gb/s$
Tasa de transferencia para el diseño completo volcando a memoria principal	$12{,}07 Gb/s$

Tabla 5.6: Rendimiento de la tasa de transferencia a través del bus PCIe.

Estos resultados empíricos se diferencian de las estimaciones de la sección 5.1. Se había estimado un rendimiento de $14,854$ Gbps frente a los $12,07$ Gbps obtenidos como la media de 100 iteraciones. La justificación a esta actividad pasa por el hecho de observar que las estimaciones se realizaron para un caso de bajo aprovechamiento de los descriptores. Es decir, únicamente se consideraban 4KB frente a los 512KB que se transmiten en la aplicación final. Esto justifica que el rendimiento real teórico sea algo superior. Que no sea exactamente el valor hallado previamente se basa en el hecho de que los elementos, aunque optimizados en la medida de lo posible, podrían no ser todo lo realmente óptimos posibles, desde que exista comunicación con otros dispositivos por el bus PCIe hasta que el core DMA no sea capaz de aprovechar todo el ancho de la línea. No se olvide que, en este caso, siempre hay datos disponibles para enviar al anfitrión por lo que las tasas obtenidas son superiores a los 10 Gbps objetivo. En el escenario real no debería observarse tal anomalía.

Para la primera prueba el diseño del driver monitoriza los tiempos empleados para realizar las operaciones y se omiten los periodos de carga asociados a solicitud de recursos y liberación de los mismos. Sin embargo, la configuración de los descriptores y la escritura de las direcciones de bus del descriptor que comienza la lista en el BAR0 de la FPGA si que quedan contemplados.

En la versión completa la caída de rendimiento se justifica por:

- La medida de las operaciones asociadas al registro del buffer (de huge pages) en el driver. Involucran una llamada IOCTL al módulo propio más una llamada mmap sobre el módulo del sistema encargado de administrar las huge pages.
- Sucesivas llamadas IOCTL al módulo propio para comprobar los datos accesibles desde el diseño de usuario para captura.
- Sincronización entre los dos hilos a nivel de usuario, donde se interponen semáforos y tanto el planificador del sistema como la carga actual influyen negativamente.

A pesar de la disminución de prestaciones obtenida, los resultados no son extremadamente desalentadores como para prever la inviabilidad del diseño en captura a 10 Gbps. Si el sistema de volcado a disco asegura una tasa superior, cabe esperar que el prototipo cumpla con las especificaciones prefijadas o el rendimiento sea cercano al deseado.

5.2.2 Comportamiento del RAID de nivel 0

Motivación de un RAID a nivel software

Se realiza una primera comprobación para medir el rendimiento real de un disco Samsung SSD 840 EVO. Para una estimación inicial de la tasas ofertadas se procede con la lectura de un flujo de datos de 50 GB sin utilizar ningún sistema de ficheros que pudieran enmascarar el comportamiento real. Se insiste en que las pruebas tienen por objetivo obtener unos valores, no necesariamente óptimos en este primer acercamiento, superiores a los 10 Gbps en lectura (se debe seleccionar, entre otros parámetros, tamaño de stripe, número de discos, raid software/raid hardware y la política de caché aplicada). La finalidad de realizar esta aproximación en lectura está motivada por la negativa a penalizar en el periodo de vida del componente evitando realizar escrituras que realmente no son necesarias. Los valores en la tabla 5.7 reflejan el comportamiento obtenido para un único disco.

Tasa de lectura mínima	543,5MB/s
Tasa de lectura máxima	564,7MB/s
Tasa de lectura media	560,6MB/s

Tabla 5.7: Rendimiento de un único disco SSD en la lectura de un fichero de 50GB (promedio 100 iteraciones).

A continuación, se utiliza la controladora hardware disponible, Adaptec RAID 6805, y se repite el experimento tras crear y configurar un RAID de nivel 0 para distintos números de discos (2 y 4). El tamaño de stripe se fija en 64 KB. Los resultados aparecen reflejados en la figura 5.3.

Los datos que aparecen en las figuras 5.3,5.4 se corresponden con la media de las velocidades máximas alcanzadas en un total de 100 iteraciones del caso de prueba. Sorprende el hecho de que el sistema no mejore con el aumento de discos. En su defecto parece decrementarse la tasa obtenida y dista de los 10 Gbps marcados como requisito. A su vez, hay que resaltar que un disco exclusivamente está asegurando una mejor actuación que el RAID. A igualdad de

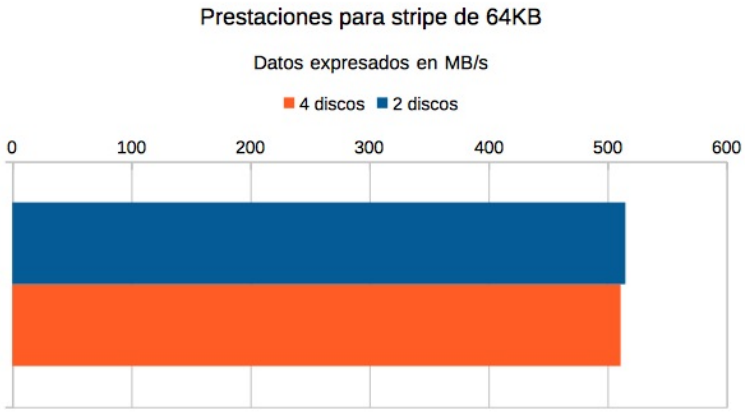


Figura 5.3: Rendimiento en lectura del RAID0 hardware a tamaño de stripe 64KB.

condiciones en tamaño de fragmento, se repite la observación para un RAID gestionado a nivel software. Los resultados son significativamente mejores como señala la figura 5.4.

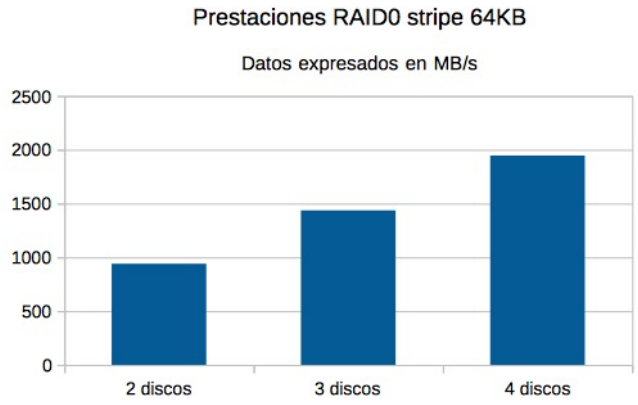


Figura 5.4: Rendimiento en lectura del RAID0 software a tamaño de stripe 64KB.

En la figura 5.3 el límite obtenido en ambas configuraciones es muy similar. Por tanto, parece que el cuello de botella no son los discos. Este hecho, a su vez, es respaldado por el rendimiento del RAID software. El limitante tampoco se trata de los buses de interconexión. La controladora hardware emplea PCIe de segunda generación (8 líneas, el rendimiento es superior al calculado en la sección 5.1 para la transferencia de datos hacia el anfitrión) y los distintos discos se conectan a la controladora mediante mini-SAS. Nótese en este punto que para la realización de las pruebas no hay posible penalización por un uso intensivo de PCIe dado que se ejecutan sobre un sistema sin apenas mayor carga de trabajo que la del propio sistema operativo. Por otra parte, la conexión mini-SAS ofrece soporte para 4 líneas SATA. Es decir, se dispone de un ancho de banda agregado de 24 Gbps para 4 discos (el límite actual para SATA es de 6 Gb/s y se conectan 4 discos, el máximo soportado por la controladora). Las pruebas apuntan a que probablemente la memoria de la controladora hardware no está dando la velocidad necesaria o la lógica de control para un RAID-0 no es óptima para la tasa esperada. Los algoritmos utilizados en ningún caso son accesibles y las especificaciones sobre los mecanismos de manejo interno de los discos son desconocidos en todo momento.

La segunda opción, una lógica de control ineficiente a la tasa de 10 Gbps, gana rigor tras

verificar que un aumento significativo del tamaño de cada pedazo provoca un aumento de la eficiencia (aumento del stripe size). Por ejemplo, para un tamaño de stripe de 1 MB se tiene una velocidad máxima de lectura de 793MB/s para un RAID-0 hardware que utiliza 4 discos de las mismas características. La controladora, en este caso, está actuando de cuello de botella y limita la viabilidad del capturador. Por tanto, se desecha la construcción de un RAID gestionado a nivel hardware.

Sin embargo, un detalle crítico que no termina de ser satisfactorio en la elección de un sistema de RAID-0 gestionado a nivel software, y que no debe ser omitido, es el efecto pernicioso que provoca en el uso de la CPU. Se intenta diseñar un sistema que en la medida de lo posible liberase de carga a la unidad de proceso de la estación de trabajo. En este caso, entran en conflicto la solución de buscar una alternativa económica que no haga un uso excesivo de CPU y que garantice las prestaciones. Bajo los resultados obtenidos se impone la necesidad de mejorar los atributos de la CPU de modo que el RAID gestionado sea capaz de alcanzar la tasa esperada de 10 Gbps.

Prestaciones. Elección de la configuración.

Una vez elegido el sistema operativo como elemento gestor del RAID faltan por especificar otros atributos. En esta ocasión se diseña un experimento consistente en la escritura de un fichero de 2GB. Se repite este proceso a lo largo de 100 iteraciones para distintos sistemas de ficheros y configuraciones de caché.

Para la ocasión se cuenta con un total de 8 discos Samsung SSD 840 EVO. Se comparan los siguientes aspectos:

- Sistema de ficheros. Se contemplan tres sistemas de ficheros de amplio uso y se evalúan los resultados con la escritura en el propio dispositivo de manera directa. No es un requisito necesario disponer de un sistema de ficheros pero si simplificará el uso por parte del usuario de la herramienta. Los sistemas sometidos a comparación son:

  – XFS [Gra14]. Es un sistema de ficheros basado en "journaling" (basado en transacciones) de alto rendimiento. XFS es especialmente eficiente en IO paralela debido a su diseño basado en grupos de asignación. Detallemos esto en más detalle, el sistema es dividido en un número de fragmentos de igual tamaño denominados Allocation Group (AG). Cada AG puede ser entendido como un sistema de ficheros individual que administra su propio espacio. Ejemplo de ello es que cada AG administra un tamaño de hasta un terabyte, cuenta con un superbloque que describe información general del sistema de ficheros, gestiona el espacio libre y es el encargado de la asignación de inodos en la memoria privada.

    Esto fomenta una alta escalabilidad del sistema (hilos con alta carga de IO, ancho de banda del sistema de ficheros y tamaño del sistema cuando abarca varios dispositivos de almacenamiento).

  – Journaled File System (JFS) fue diseñado con la idea de asentarse en estaciones de trabajo de alto rendimiento. Entre las características más reseñables del sistema se destaca la alta rapidez de recuperación en caso de caída del sistema y la tendencia a utilizar menos CPU que otras alternativas [Han06]. Sin embargo, también se ha apreciado una disminución de rendimiento al trabajar con numerosos ficheros [Han06] [Com03].

  – Ext4 es el sistema de ficheros más extendido en sistemas operativos linux. Una evolución de Ext3, con modificaciones en las estructuras de datos ofertando una mejora en rendimiento, fiabilidad y características.

Desactivando todas las políticas de caché, el comportamiento de un único disco para el tamaño de stripe por defecto (64KB), es muy similar y el orden de la penalización por usar un sistema de ficheros es aproximadamente del 8,9 %.

Para verificar la escalabilidad de cada uno de los sistemas se repite el procedimiento para 4 y 8 discos,dado que para un único disco no hay ningún factor discriminante tras la observación de los datos iniciales. En vista de las muestras obtenidas empíricamente, se detecta una peor escalabilidad bajo el sistema de ficheros JFS. Ext4 y XFS parecen mantener unos resultados bastante similares a lo largo de todas las pruebas, aportando este último unas tasas de escritura más elevadas ligeramente.

- Impacto de la caché de los discos en el rendimiento. Como segundo aspecto de configuración se evalúa la posibilidad de habilitar la caché propia de cada uno de los dispositivos. Su configuración se puede realizar mediante la herramienta hdparm (ver parámetro -W de la aplicación). Una unidad de almacenamiento que provee de caché en escritura, en caso de activación, ofrecerá un mayor rendimiento a cambio de un aumento de la posibilidad de pérdida de datos ante un fallo de alimentación de la unidad.

  Un disco duro con la caché de escritura habilitada responde con un mensaje de finalización a un comando de escritura inmediatamente después de la petición de escritura y antes de que los datos sean realmente escritos. Esta respuesta instantánea permite al anfitrión proceder con la siguiente petición de escritura garantizando teóricamente un mejor rendimiento en general. Nuevamente, recalcar que esta ganancia se debe a un aumento del riesgo de pérdida de datos en caso de fallo en la alimentación. Si un disco es apagado antes de que se complete la operación actual los datos de la memoria caché serán perdidos para siempre.

  En el índice anterior las pruebas reflejaban el comportamiento cuando se deshabilita esta opción. En esta ocasión se analiza la mejora real de activar la caché de los discos. La aceleración obtenida es cercana al 4x en los dos primeros casos (algo inferior en el tercero).

  Como consideración general, en un RAID0 la redundancia aplicada es nula y el fallo en cualquiera de los discos podría comprometer la integridad de todos los datos almacenados. Sin embargo, deshabilitar esta opción en los discos como medida para asegurar la integridad de la información no se puede contemplar como una solución completamente satisfactoria. El sistema no se asegura en ningún caso frente a la corrupción de los datos o los riesgos asociados a la escritura en disco. Es por ello, que en en la búsqueda de un sistema de almacenamiento de gran tasa de escritura, se prima la velocidad frente a la fiabilidad y robustez de los datos con las posibles consecuencias que esta decisión de diseño pudiera acarrear.

- Importancia de las políticas de caché. Para completar el análisis se comparan las políticas de caché write back (WB) y write through (WT). Se configuran a nivel software durante la creación del RAID (ver comando mdadm y parámetro layout). Los distintos métodos son exclusivos (no se aplican simultáneamente) y definen las siguientes características:

  - WB. Las modificaciones a los datos en la caché no son copiadas a la fuente hasta que es absolutamente necesario.
  - WT. Las modificaciones a los datos en la caché son copiadas a la fuente a la vez que son modificadas en memoria caché.
  - Escritura directa (IO direct). No se hace uso de memoria caché y los datos son directamente escritos en la unidad de almacenamiento.

Nuevamente se realizan las pruebas para distinto número de discos en el RAID. Parece que la opción de no volcar los datos hasta que es realmente necesario es la que mejor resultados ofrece. La opción de volcar paralelamente (WT) ofrece unos resultados similares a la opción de escritura directa como era de esperar. El hecho de que WT ofrezca mejores resultados es fácilmente justificable ya que el acceso a memoria principal, típicamente será de varios órdenes de magnitud inferior al de escritura en los dispositivos externos. El tipo de escritura realizada, que puede considerarse secuencial, la asociada al fichero de 2GB, beneficia que la actividad de no volcar los datos hasta última instancia sea ventajosa. No se realizan escrituras de un mismo bloque un número innecesario de veces.

Otros experimentos, donde se forzase a sobrescribir la región de memoria previamente escrita (escritura inicial secuencial sucedida de sobrescrituras no secuenciales en los mismos sectores de unos pocos bytes) posiblemente mostrasen otro tipo de comportamientos. En cualquier caso, el objeto de la aplicación será volcar una información a regiones de memoria, típicamente contiguas, si el disco lo permite (no olvidemos que en caso de fragmentación no hay tampoco penalización por el tipo de unidad de almacenamiento).

Conclusiones finales sobre sistema de almacenamiento

A lo largo de la presente sección se han evaluado y comparado distintas alternativas para la administración de un RAID de nivel 0. Se compara su gestión desde una controladora a nivel hardware frente a un modelo puramente software, ofreciendo este último unas prestaciones más elevadas.

Ligado a la gestión de los archivos se evalúa la escalabilidad y rendimiento de tres sistemas de ficheros diferentes: XFS, JFS y Ext4. Se verifica que el primero parece obtener unos mejores comportamientos globales y se decide aplicar por las numerosas ventajas que desde el punto de vista del usuario ofrece tener un sistema común y estándar frente a una alternativa propia que se pudiera desarrollar.

Se analizan distintas políticas de caché para determinar que las opciones que mejor se ajustan al problema propuesto pasan por habilitar la caché de los discos y utilizar write back. Se prueba a aumentar el tamaño de "stripe". A lo largo de todas las figuras y configuraciones se mantenía fijo a 64KB. Incrementando este valor a 512KB, se consiguen las siguientes medidas finales, satisfactorias para el propósito de escribir a una tasa que asegure, al menos, 10 Gbps (los datos exactos se muestran en la tabla 5.8).

Tasa de lectura mínima	2327,12MB/s
Tasa de lectura máxima	2388,33MB/s
Tasa de lectura media	2410,14MB/s

Tabla 5.8: Rendimiento final del RAID de 8 discos SSD en la escritura de un fichero de 2GB bajo el sistema de ficheros XFS.

A raíz de los resultados desprendidos del aumento de rendimiento debido al cambio del tamaño de "stripe", se comprueba que el uso de CPU no es despreciable por el RAID. El aumentar el tamaño de "stripe" implica una menor carga por parte de la lógica de control y se obtiene una aceleración superior al 1.5x. Con esta configuración se asegura la recepción exitosa a la tasa objetivo de 10 Gbps y permite margen para su evolución hacia sistemas más ambiciosos. No hay que obviar que la tasa real obtenida no está escalando linealmente con el número de discos y que los resultados teóricos distan de los empíricos. Si se desea mayor tasa, añadir nuevos discos no será seguramente una buena opción.

5.2.3 Rendimiento general del capturador de tráfico

Para medir las prestaciones generales del programa creado se exige de la instalación de la máquina en un entorno donde se pueda generar tráfico a la tasa indicada. Para tal efecto se configura un sistema basado en la misma arquitectura que la del capturador presentado. Se lee de una traza en el formato de la sección 4.2.1 y se envían a través del bus PCIe hacia la tarjeta, donde los datos se transmiten a través de XGMII al core XAUI de Xilinx y son finalmente puestos en la red. Es la aplicación presentada en la sección 4.4.4.

En el otro extremo, a la escucha se encuentra el prototipo de captura. Se mide el tiempo que tarda el diseño de usuario desde que comunica que desea empezar a capturar el primer paquete hasta que se capturan 20GB. El generador de tráfico está enviando datos en bucle de modo que siempre hay tráfico disponible para la escucha.

Se repite la prueba para los tamaños de paquete extremos, 64B y 1518B. Es decir, inicialmente se envían paquetes de tamaño 64B hasta que la prueba es completada. Se repite el proceso para 1518B. Se repite el experimento 10 veces y se promedian los resultados. Las medidas obtenidas se reflejan en la tabla 5.9 para una primera versión que contabiliza el número de bytes recibidos y divide entre el tiempo empleado (normalizando para que las unidades sean expresadas en Gbps).

Tasa de lectura media para 64B	8,56408Gb/s
Tasa de lectura media para 1518B	9,92185Gb/s

Tabla 5.9: Rendimiento de captura para distintos tamaños de ficheros teniendo en cuenta el número de bytes volcados a disco.

Los datos parecen ser satisfactorios para el caso de tramas de tamaño de 1518B. Sin embargo, se nota una diferencia notoria para el primer caso. En este punto, es conveniente destacar que las interfaces aseguran la transferencia a 10 Gbps pero que en tales comunicaciones no todos los datos transferidos son bytes válidos. Existen señales de control, de redundancia de datos, que deben ser contabilizadas también para realizar un análisis justo de la tasa. Observamos que:

- Cada paquete es precedido por un preámbulo de 8 bytes (protocolo XGMII).
- Cada paquete al terminar incorpora una secuencia de terminación de 4 bytes. El generador de tráfico hace uso de la técnica conocida como deficit idle count de manera que se ajusta para que el tamaño de esta palabra y el tiempo entre paquetes sea igual a 12 bytes en promedio [Spi12]. Es decir, entre dos paquetes consecutivos existe un periodo de inactividad equivalente al tiempo transcurrido en enviar 3 palabras de 4 bytes.

En total se cuenta con una penalización de 20 bytes. Teniendo en consideración estos factores se tiene que en la interfaz de escucha la tasa de llegada de datos se sitúa en los valores de la tabla 5.10. Se finaliza, afirmando que el diseño presentado es capaz de realizar la captura en redes multigigabit ethernet a una tasa de 10 Gbps sin pérdida de paquetes.

Tasa de lectura media para 64B	9,9999002Gb/s
Tasa de lectura media para 1518B	9,9998791Gb/s

Tabla 5.10: Rendimiento de captura para distintos tamaños de ficheros teniendo en cuenta los bytes de control.

5.3 Resumen: prestaciones y evaluación

En esta sección se han analizado los principales elementos de la cadena en serie que conforman el capturador. Se detectan como principales cuellos de botella el acceso a memoria y a las unidades de almacenamiento. El bus PCIe ofrece tasa suficiente para la aplicación deseada a tasas de 10 Gbps pero debe asegurarse que hay un tratamiento óptimo de los datos porque el margen está bastante afinado a la tasa deseable. Para solucionar los problemas de acceso a memoria dentro del diseño hardware, se utilizan memorias block RAMs con capacidad para leer y escribir datos en un único ciclo de reloj. Para evitar una penalización adicional por el acceso a la memoria principal del sistema, driver y diseño de usuario comparten la misma región. Al no realizar copias innecesarias, técnica "zero-copy", el único cuello de botella limitante es el RAID de discos SSD. Un total de 8 discos aseguran la escritura a 10 Gbps solventando el problema. Permiten además la instalación de un sistema de ficheros como es XFS. La escalabilidad del sistema de almacenamiento debe ser tenida en cuenta para aplicaciones basadas en este diseño ya que la linealidad de las operaciones al añadir nuevos discos no está asegurada.

6 Conclusiones y trabajo futuro

A lo largo de este trabajo fin de grado se han expuesto los puntos clave de un capturador de tráfico a 10 Gbps económico y escalable. En la sección 4.1 se informa sobre el coste aproximado de cada uno de los elementos del equipo. Los componentes básicos se sitúan en torno a $4092(≈ 3009€), de los cuales más de una tercera parte se corresponde con una tarjeta NetFPGA cuya funcionalidad es expansible mucho más allá de la captura de tráfico.

Mención separada requiere la implantación del sistema en otros entornos. El sistema desarrollado es altamente dependiente del sistema operativo. Se ha probado su funcionalidad en versiones del kernel comprendidas entre la 2.6.32 y la 3.13. No se asegura el correcto funcionamiento en versiones diferentes aunque se pretende seguir dando soporte a la aplicación (ver trabajo futuro).

Respecto a otras limitaciones físicas, el mayor cuello de botella detectado es el dispositivo de almacenamiento. Para lograr el resultado esperado ha sido necesario recurrir a un total de 8 discos duros de estado sólido aunque según las especificaciones del fabricante, y suponiendo que no hay penalización por el uso de un RAID de nivel 0, con reducir el número a la mitad hubiera sido suficiente. Sin embargo, al disponer de un número mayor de discos, el espacio disponible para la captura de datos es superior, lo que ha permitido realizar un monitorizado de la red durante periodos mayores de tiempo.

La tasa de captura es sostenible en líneas generales y pese al requisito inicial de liberar de carga de trabajo a la CPU, esta restricción no parece haberse cumplido. El problema no se corresponde con un diseño de usuario ineficiente o una sobrecarga por parte del módulo kernel. La justificación de la ocupación general del sistema está apuntada por la elección de gestionar el RAID a nivel software. Esta operación somete a un core de la CPU a estar trabajando al máximo de su capacidad durante los tiempos de escritura. Otros tres cores quedan prácticamente disponibles en su totalidad el resto del tiempo. Las principales distinciones que este capturador ofrece frente a otras alternativas mencionadas en la sección 2 son:

- Captura real a 10 Gbps.
- Herramienta capaz de capturar al nivel de la capa física en el modelo OSI, protocolo XGMII, ofreciendo una resolución precisa en la llegada de paquetes y recogiendo hasta las trazas con errores de forma.
- Transferencia de datos mediante DMA desde la memoria de la tarjeta FPGA hasta el anfitrión.
- Gestión de interrupciones MSI para evitar que la aplicación haga uso de técnicas como polling sobre la placa FPGA con el respectivo consumo asociado.
- Uso de páginas de tamaño no estándar del sistema.
- Creación de un anillo de descriptores que transmiten la información sin necesidad de intervención de la CPU gracias al uso de la técnica DMA "scatter-gather".

En líneas académicas, el trabajo realizado ha sido muy enriquecedor gracias al contacto con distintas materias. La mayor parte del desarrollo guarda relación con asignaturas cursadas durante la carrera pero ha requerido una profunda labor de investigación y aprendizaje. De forma global se puede enmarcar el producto en las ramas de:

- **Arquitectura de computadores**. Una parte no despreciable del proyecto es un diseño hardware. La destreza del tutor en este ámbito ha sido un apoyo clave en la realización.

Los errores cometidos en hardware son, sin lugar a dudas, mucho más difícilmente detectables que los cometidos en software. El generar un código limpio, que cumpliera las restricciones de tiempo establecidas ha sido otro de los grandes retos a afrontar. La documentación facilitada por Xilinx [Xila] es la guía general para aprender a hacer un buen uso de los cores que la herramienta ISE trae integrados.

- **Sistemas operativos**. Para la generación de un driver [JC06] ha supuesto un punto de partida. No obstante, la antigüedad del texto hace que sirva más como una referencia que como un modelo a seguir. En ese sentido, la lectura de los propios documentos, así como el código, suministrados en la descarga del kernel linux suponen el siguiente paso. En ningún caso el kernel de linux cuenta con un depurador integrado que facilite la detección de errores. Linus Torvalds, creador del kernel, opina que un depurador no ayuda a detectar los fallos, únicamente sirve como método para buscar parches a los errores sin localizarlo realmente.

- **Redes de telecomunicaciones**. Se ha implementado la capa de nivel físico en el modelo OSI. Aunque en esta parte más técnica no se ha entrado en profundo detalle, el poseer el concepto general ayuda a que la documentación sea entendible. El referente es el IEEE 802.3 [Soc12].

6.1 Trabajo futuro

A pesar del éxito logrado en la realización de la aplicación, un abanico muy amplio de posibilidades se extiende a partir de este prototipo.

- **Liberación del código bajo una licencia libre**. Uno de los principales objetivos en el momento de redacción del documento es la realización de casos de pruebas exhaustivos del sistema, de modo que una vez asegurada la correctitud del programa se pueda poner a disposición de cualquier tercero. Es primordial asegurarse de la estabilidad en otros entornos distintos del especificado (equipo físico). Dado que se utiliza un driver propio, una gestión inadecuada podría bloquear los recursos del anfitrión y es algo que debe ser revisado concienzudamente.

- **Librería para el núcleo del sistema con soporte para huge pages**. La aplicación se ve obligada a realizar la petición de memoria asociada a las huge pages desde el nivel de usuario para su posterior comunicación al driver. Esta técnica podría ahorrarse si el sistema operativo dispusiera de las funciones oportunas para la solicitud de memoria de las huge pages desde el propio kernel. Dado que linux es un sistema de código libre, la implementación de tal funcionalidad es viable y ayudaría a otros desarrolladores que necesitasen implementar esta misma funcionalidad.

- **Realización de pruebas de rendimiento del sistema de almacenamiento**. El uso de una controladora hardware, adaptada a discos duros de estado sólido, podría acarrear una mejora de los resultados. Una buena opción sería evaluar si el sobrecoste que impone la adquisición de una controladora de este tipo puede ser suplido con la reducción del número de discos que deban usarse en la aplicación.

- **Diseño de capturador a 40 Gbps**. Sentadas las bases de la arquitectura, cabría esperar una escalabilidad del sistema bajo los siguientes supuestos:

  – Sustitución de la tarjeta Virtex 5 de Xilinx (tarjeta NetFPGA) por una placa Virtex 7 de Xilinx con interfaces de 40 Gbps como el modelo VC 709.

  - Aprovechamiento óptimo del bus PCIe de tercera generación. A esta tasa sería inviable utilizar el bus de segunda generación.
  - El sistema de almacenamiento presentado en esta ocasión raramente ofrecerá el rendimiento idóneo. Para un diseño donde únicamente se monitoricen flujos, en lugar de paquetes individuales, se podría alcanzar la velocidad.

- **Generador de tráfico a una tasa multigigabit especificada (≤10 Gbps)**. La arquitectura presentada es completamente portable a una utilidad cuya funcionalidad sea la inversa como se ha visto en la sección 4.4.4. Es decir, cuya misión sea la de enviar tráfico a partir de un fichero en disco. Se ha analizado que en cuanto al equipo físico ninguna modificación sería necesaria. Relativo a los diseños, un nuevo propósito del módulo "app" en el diseño hardware (ver figura 4.17) ha sido necesario implementar. El módulo "app" puede optimizarse para realizar un cálculo del interframe gap entre paquetes de manera dinámica. De este modo podría configurarse a través de los registros de control la velocidad deseada dejando al diseño hardware la reproducción a dicha tasa sin necesidad de modificar el fichero original.

- **Soporte para múltiples interfaces a 10 Gbps**. El capturador desarrollado únicamente cuenta con soporte para una única interfaz. Un total de tres interfaces más no se están gestionando en la actualidad. En captura, un cuello de botella de difícil solución (queriendo mantener la premisa de construir un sistema económico) es el dispositivo de almacenamiento. Sin embargo, pensando en un generador de tráfico, que una misma máquina fuera capaz de reproducir tráfico hacia cuatro estaciones de trabajo simultáneamente, a la tasa de 10 Gbps, puede garantizar un buen sistema para la realización de pruebas en entornos distribuidos (igual carga a un total de 4 nodos).

- **Soporte para captura y reproducción a 10 Gbps**. Simplificando el sistema de escucha para que no sea necesario el volcado a disco, una utilidad que escuchase en la red, transfiriese los datos al sistema anfitrión y los devolviese de nuevo a la red puede tener varias aplicaciones a considerar:

  - Elemento de monitorización del enlace. La más simple de las funcionalidades puede ser medir la ocupación de la conexión, distribución del tráfico en el tiempo o control de las máquinas que más utilizan dicho enlace. Para ello es necesario que el programa de usuario que aplique los filtros sobre los paquetes sea suficientemente rápido como para asegurar la tasa.
  - Elemento de filtrado. En relación al apartado anterior, puede pensarse en un sistema de captura, filtrado y reproducción. Todos los paquetes de la red pueden ser monitorizados. Aquellos que cumplan una condición prefijada se transfieren al anfitrión (por ejemplo, paquetes HTTP). Finalmente, el usuario puede aplicar restricciones sobre los paquetes (para completar la situación anterior basta pensar en el bloqueo de una lista de URLs no permitidas). La información no deseada nunca alcanzaría el otro extremo del enlace y los filtros se elaboran en cualquier lenguaje de programación convencional.

Bibliografía

[ARM10] ARM. Amba axi4-stream protocol specification. 2010.

[Cen14] CentOS. Página oficial de la distribución gnu/linux centos, 2014. URL: https://www.centos.org/.

[Com03] Netnation Communications. Linux file system benchmarks. 2003. URL: http://fsbench.netnation.com/.

[Cor12a] Xilinx Corp. Logicore ip block memory generator v7.2 product guide, 2012.

[Cor12b] Xilinx Corp. Logicore ip fifo generator v9.3, product guide, December 2012.

[Cor12c] Xilinx Corp. Xilinx pg053 logicore ip xaui v10.4, product guide. 2012.

[Cor14] Xilinx Corp. Software defined specification environment for networking, 2014.

[For00] Task Force. Xaui/xgxs proposal. 2000.

[Glo] HiTech Global. Xilinx virtex®5 tx240t pci express and 40 gig sfp+ development platform. URL: http://www.hitechglobal.com/Boards/PCIExpress_SFP+.htm.

[Gra14] Silicon Graphics. Xfs file system official web page, 2014. URL: http://xfs.org.

[Han06] Hansivers. Filesystems (ext3, reiser, xfs, jfs) comparison on debian etch. 2006. URL: http://www.debian-administration.org/articles/388.

[Int] Intel. Intel® dpdk, set of libraries and drivers for fast packet processing on x86 platforms. URL: http://dpdk.org/.

[Int06] Intel. White paper intel i/o acceleration technology. 2006.

[JC06] Greg Kroah-Hartman Jonathan Corbet, Alessandro Rubini. Linux device drivers. 2006.

[Log] Northwest Logic. Northwest logic pcie core. URL: http://nwlogic.com/.

[Log14] Northwest Logic. Expresso dma core. 2014.

[nto14] ntop. n2disk project official web page. 2014. URL: http://www.ntop.org/products/n2disk/.

[Ora14] Oracle. Mysql 5.7 reference manual. 2014.

[PS05] PCI-SIG. Pci express base 1.1 specification, 2005.

[PS10] PCI-SIG. Pci express base specification revision 3.0, 2010.

[RL13] Iván Gonzalez Rafael Leira. Clasificación de flujos en 10g ethernet mediante intel dpdk y gpus. 2013.

[Sam13] Samsung. Samsung ssd 840 evo data sheet, 2013.

[SH11] KyoungSoo Park-Sue Moon Sangjin Han, Keon Jang. Packetshader, a gpu-accelerated software router, 2011.

[Soc02] IEEE Computer Society. Ieee standard for ethernet 802.3ae. 2002.

[Soc12] IEEE Computer Society. Ieee standard for ethernet 802.3. 2012. URL: http://standards.ieee.org/about/get/802/802.3.html.

[Spi12] Spirent. How to test 10 gigabit ethernet performance, March 2012.

[Tec03] Information Society Technologies. Ist scampi project. 2003. URL: http://www.ist-scampi.org/.

[Tec05] Information Society Technologies. Ist lobster project. 2005. URL: http://www.ist-lobster.org/.

[Uni] Standford University. Netfga project. URL: https://github.com/NetFPGA,http://netfpga.org.

[Wha13] Edward Whalen. How to configure x86 memory performance for large databases. 2013.

[Xila] Xilinx. Xilinx official web page. URL: www.xilinx.com.

[Xilb] Xillybus. An fpga ip core for easy dma over pcie with windows and linux. URL: http://xillybus.com.

[Xil05] Xilinx. Using block ram in spartan 3 generation fpgas. 2005.

A Cuestiones pertinentes al sistema operativo

Compilación de un kernel personalizado.

En muchas ocasiones es beneficioso contar con las novedades que un kernel más reciente puede aportar tanto a la estabilidad del sistema como en las utilidades incorporadas. La aplicación desarrollada explota numerosos aspectos del sistema operativo, directa o indirectamente. Para motivar la actualización a un kernel mayor se plantea el siguiente ejemplo.

Se ha observado que en el manejo de un RAID de discos de estado sólido no está implementado el soporte para TRIM (mecanismo de comunicación de sectores libres entre el sistema operativo y un disco SSD). Aunque tal funcionalidad está implementada para el manejo de un único disco, para un RAID no se incorporó dicho mecanismo hasta la versión 3.7 del kernel. La distribución CentOS, en su versión 6.5, trae un kernel por defecto 2.6.32-431. Como podría ser conveniente disfrutar de las novedades de un nuevo núcleo, que añadiese esta funcionalidad con su mejora en rendimiento asociada, en este apéndice se detallan los pasos a seguir para su instalación.

- El primer paso, consiste en descargar la versión deseada del kernel desde la web `www.kernel.org`.
- Tras descomprimir el fichero y ubicarse con una terminal en directorio creado, se procede a seleccionar los módulos que formarán parte del sistema. Para la selección manual de los elementos se ejecuta el comando "make menuconfig". Si se prefiere una interfaz visual "make xconfig" ofrece las mismas opciones.

  Para el capturador de tráfico es necesario habilitar:

  - Módulos correspondientes al hardware de la máquina. Si disponemos del fichero .config de un sistema creado anteriormente puede ser interesante usar el comando "make oldconfig" para evitar marcar los módulos nuevamente. Otra opción sería utilizar el comando "make defconfig" para generar una configuración básica por defecto que pudiera funcionar según qué tipo de máquina se esté empleando.
  - Huge Pages. Es necesario marcar la opción: File systems -> Pseudo filesystems -> HugeTLB file system support.
  - XFS. Se habilita para su soporte el módulo: File systems -> XFS filesystem support.
  - RAID. Si únicamente se usará un RAID de nivel 0 basta con activar:

  Device drivers -> Multiple devices driver support (RAID and LVM) -> RAID support.

  Device drivers -> Multiple devices driver support (RAID and LVM) -> RAID-0 (striping) mode.

- Finalmente falta compilar el nuevo sistema mediante los comandos del cuadro A.1.

```
1  make -j9
2  make modules
3  make modules_install
4  make install
```

Cuadro A.1: Pasos convencionales para la compilación de un kernel personalizado.

Creación de un RAID a nivel software.

Para la gestión y configuración de un RAID a nivel software bajo un sistema GNU/linux, la herramienta empleada es mdadm, que implementa una capa de abstracción sobre los dispositivos. Una vez realizada la instalación inicial del RAID, el usuario verá la unidad de almacenamiento integrada por los distintos discos como un único dispositivo adicional. En los sucesivos ejemplos esta unidad es referenciada como /dev/md0. Mdadm se encargará de aplicar la lógica necesaria en cada transferencia que se realice sobre el citado dispositivo. El primer paso para crear un RAID desde la nada es borrar la información que pudiera contener cada uno de los discos asociada a una configuración previa. Para realizar el borrado basta con seguir las instrucciones del cuadro A.2.

```
1  #mdadm --zero-superblock /dev/<drive>
```

Cuadro A.2: Secure wiped de un disco.

Posteriormente, se procede con la creación del RAID propiamente dicho. Este es el momento en el que el número de discos, nivel del RAID o parámetros propios del nivel son configurados. Para el hipotético caso de la creación de un RAID de nivel 0, con tamaño de stripe 64 KB (es decir, cada fichero se fragmentará en pedazos de este tamaño que será repartidos entre los diferentes discos) y dos discos, la sentencia queda contemplada en el cuadro A.3.

```
1  #mdadm --create --level=0 --chunk=64 --raid-devices=2 /dev/md0 /dev/
       sdb /dev/sdc
```

Cuadro A.3: Creación de un raid de nivel 0 con tamaño de stripe 64KB.

Es conveniente resaltar que los parámetros establecidos en momento de creación no pueden ser modificados posteriormente sin poner en peligro la integridad de los datos. Para asegurar una experiencia más agradable al usuario falta dotar de un sistema de ficheros a la unidad. Según el formato usado es conveniente calcular ciertos parámetros que mejorarán notoriamente el rendimiento. Por ejemplo, para el sistema de ficheros ext4:

1. stride
$$\text{stride} = \frac{\text{chunk size}}{\text{block size}} = \frac{64KB}{4KB} = 16$$
2. stripe-width
$$\text{stripe-width} = \text{Number of disks} * \text{stride} = 2 * 16 = 32$$

La acción de dotar de formato a la unidad virtual pasa por ejecutar la instrucción del cuadro A.4.

```
1  mkfs.ext4 -b 4096 -E stride=16,stripe-width=32 /dev/md0
```

Cuadro A.4: Sistema de ficheros Ext4 en dispositivo RAID.

Finalmente, se almacenan los ficheros de configuración para que en sucesivos reinicios el sistema sea capaz de detectar el dispositivo (cuadro A.5).

```
1  # mdadm --detail --scan >> /etc/mdadm.conf
```

Cuadro A.5: Fichero configuración RAID.

Fichero virtual gestionado por el módulo kernel.

El usuario final es capaz de comunicarse con un módulo del kernel o, también denominados drivers, a través de dos formas diferenciadas principalmente:

- Mediante la creación de un dispositivo de caracteres ("char device"). Se creará un dispositivo virtual bajo la carpeta "/dev", gestionado por el módulo del sistema. A través de éste se permite interactuar y modificar la actividad del módulo u obtener información. Ejemplos típicos de estas acciones son la lectura/escritura sobre el dispositivo aunque el driver será el encargado de manejar realmente los datos y definirá el concepto de estas acciones sobre el dispositivo. Ejemplos de dispositivos especiales son "/dev/zero" o "/dev/null". En el primer caso la lectura consiste en devolver un stream de ceros de tamaño igual a la cantidad solicitada; en el segundo, cualquier escritura será desechada. Están definiendo operaciones de lectura/escritura pero divergen del concepto que se puede tener asociado a la lectura, por ejemplo, de un fichero de datos convencional.

  Sin embargo, estos dispositivos no se limitan a la especificación de funciones de IO. La posibilidad de ejecutar una función ante una llamada a la función mmap sobre el dispositivo (comúnmente utilizado para compartir memoria del kernel al diseño de usuario) también se puede implementar mediante un char device. El uso de llamadas personalizadas al driver mediante la función ioctl es también soportado. La lista es amplia y queda reflejada en el cuadro A.8 aunque en cualquier caso estas operaciones son las más representativas (apertura, lectura, escritura, mmap y ioctl).

- A lo largo de la evolución del sistema operativo linux se ha expandido la tendencia de agrupar los mecanismos de comunicación con el sistema operativo bajo la carpeta "/proc". Su funcionalidad puede ser reemplazada por un char device en la práctica totalidad de las ocasiones pero conceptualmente se pretende aislar los dispositivos físicos, como unidades de almacenamiento, de los dispositivos virtuales creados por los módulos del sistema. En el caso del driver desarrollado, se implementa la posibilidad de recibir información del diseño hardware mediante el fichero "/proc/nfp/nfp_report". Es decir, la lectura sobre el fichero mencionado responderá con un array de caracteres informando del estado actual en lenguaje natural.

En un sistema en continua evolución, los prototipos de funciones varían a lo largo de su desarrollo. Este es el caso de la manera de registrar un fichero virtual bajo "/proc". El código para versiones del kernel inferiores a la 3.10 aparece en el cuadro A.6. Para versiones posteriores la manera de realizar la creación del fichero queda definida en A.7. El módulo desarrollado contempla ambas posibilidades y compilará sin necesidad de cambio en ambos entornos.

```
1  /* Prototypes of read/write functions. */
2  static int nfp_proc_read (char *buffer, char **start,
3                            off_t off, int count,
4                            int *eof, void *data);
5
6  static int nfp_proc_write (struct file *file,
7                             const char *buffer,
8                             unsigned long count,
9                             void *data);
10
11 /* General structs. */
12 static struct proc_dir_entry *nfp_dir,  /**< The folder */
13              *nfp_proc_file;           /**< The proc file  */
14
15 /* Code for registering the virtual file. */
16 nfp_dir = proc_mkdir ("nfp", NULL);
17
18 nfp_proc_file = create_proc_entry ("nfp_report", 0644, nfp_dir);
19 nfp_proc_file->data = NULL;
20 nfp_proc_file->read_proc  = nfp_proc_read;
21 nfp_proc_file->write_proc = nfp_proc_write;
```

Cuadro A.6: Diferencias entre distintas versiones del kernel para registrar un fichero en "/proc". Versiones inferiores a la 3.10.

```
1  /* Prototypes of read/write functions. */
2  static ssize_t nfp_proc_read (struct file *filp,
3                                char __user  *buffer,
4                                size_t count,
5                                loff_t *offp);
6
7
8  static ssize_t nfp_proc_write (struct file *filp,
9                                 const char __user *buffer,
10                                size_t count,
11                                loff_t *offp);
12
13
14 struct file_operations proc_fops = {
15   .owner      = THIS_MODULE,
16   .read       = nfp_proc_read,
17   .write      = nfp_proc_write
18 }; /**< The file operations struct (pointer to functions) that will be
       invoked under reads of the proc file. */
19
20 /* General structs. */
21 static struct proc_dir_entry *nfp_dir,  /**< The folder */
22              *nfp_proc_file;           /**< The proc file  */
23
24 /* Code for registering the virtual file. */
25 nfp_dir = proc_mkdir ("nfp", NULL);
26 nfp_proc_file = proc_create ("nfp_report", 0644, nfp_dir, &proc_fops);
```

Cuadro A.7: Diferencias entre distintas versiones del kernel para registrar un fichero en "/proc".Versiones mayores o iguales a la 3.10.

Como se puede observar, el principal cambio es la introducción de una estructura "file operations" para definir las funciones invocadas ante las distintas operaciones. En este caso únicamente se ha definido el comportamiento para escritura/lectura pero la lista es más amplia (ver cuadro A.8).

La utilización de esta estructura, utilizada previamente para registrar un "char device" (entre otros dispositivos), obliga a modificar los prototipos de las funciones para comunicarse con un fichero bajo "/proc" de modo que se pueda ajustar a la interfaz común previamente establecida. Se persigue mantener la compatibilidad con un mayor número de aplicaciones pero, a su vez, intentar estandarizar la manera de registrar un punto de comunicación con el usuario, ya sea un char device o un fichero virtual.

```
1  struct file_operations {
2    struct module *owner;
3    loff_t (*llseek) (struct file *, loff_t, int);
4    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
5    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t
         *);
6    ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned
         long, loff_t);
7    ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned
         long, loff_t);
8    int (*iterate) (struct file *, struct dir_context *);
9    unsigned int (*poll) (struct file *, struct poll_table_struct *);
10   long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
11   long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
12   int (*mmap) (struct file *, struct vm_area_struct *);
13   int (*open) (struct inode *, struct file *);
14   int (*flush) (struct file *, fl_owner_t id);
15   int (*release) (struct inode *, struct file *);
16   int (*fsync) (struct file *, loff_t, loff_t, int datasync);
17   int (*aio_fsync) (struct kiocb *, int datasync);
18   int (*fasync) (int, struct file *, int);
19   int (*lock) (struct file *, int, struct file_lock *);
20   ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t
         *, int);
21   unsigned long (*get_unmapped_area)(struct file *, unsigned long,
         unsigned long, unsigned long, unsigned long);
22   int (*check_flags)(int);
23   int (*flock) (struct file *, int, struct file_lock *);
24   ssize_t (*splice_write)(struct pipe_inode_info *, struct file *,
         loff_t *, size_t, unsigned int);
25   ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info
         *, size_t, unsigned int);
26   int (*setlease)(struct file *, long, struct file_lock **);
27   long (*fallocate)(struct file *file, int mode, loff_t offset,
28                     loff_t len);
29   int (*show_fdinfo)(struct seq_file *m, struct file *f);
30 };
```

Cuadro A.8: Estructura file_operations en la versión 3.13 del kernel.

Implementación de la rutina IOCTL en un módulo del kernel

Un programa de usuario puede interactuar con un driver a través de un fichero de caracteres como se ha visto a lo largo del apéndice A. Para realizar tal operación es necesario registrar

una serie de constantes que identificarán de manera unívoca a un comando IOCTL y su invocación desde un programa a nivel de usuario. En el cuadro A.9 aparece la declaración de las operaciones en un fichero de cabeceras.

```
1  #define IOCTL_MAGIC_NUMBER 'j'
2
3  #define NFPIOC_S2C_DMA              _IOR(IOCTL_MAGIC_NUMBER, 1, struct
       dma_transfer)
4  #define NFPIOC_C2S_DMA              _IOW(IOCTL_MAGIC_NUMBER, 2, struct
       dma_transfer)
5  #define NFPIOC_UNREGISTER_BUFFER _IO(IOCTL_MAGIC_NUMBER, 3)
6  .
7  .
8  .
```

Cuadro A.9: Declaración de las constantes que identificarán una operación IOCTL.

El primer parámetro que reciben las macros _IOR, _IOW, _IO es un valor de 8 bits que identifica al módulo. Es necesario revisar que no esté siendo utilizado por otro driver del sistema. La lista puede encontrarse en el fichero magic_numbers.txt del kernel Linux. El segundo argumento es el número de operación dentro del propio módulo. Así pues, el driver desarrollado soporta distintas operaciones que tendrán identificadores diferenciados. Finalmente, las macros _IOR, _IOW, _IO indican si se espera que haya un argumento extra y si se lee/escribe sobre el mismo. _IO no espera ningún argumento extra, _IOR espera un argumento más que será únicamente leído y _IOW involucra un tercer argumento que puede ser modificado.

En el nivel de módulo, es necesario que en el instante de registrar el dispositivo de caracteres se especifique el campo unlocked_ioctl y compat_ioctl (ver cuadro A.8) en la estructura file_operations asociada. Estos campos son punteros a funciones como las mostradas en el cuadro A.10. En el prototipo de la función cobra especial relevancia el último argumento que identifica el parámetro comunicado por el diseño en la llamada a la función IOCTL (a nivel de usuario). El segundo campo, "cmd", permite distinguir entre los distintos comandos implementados.

```
1  long nfp_ioctl (struct file *f, unsigned int cmd, unsigned long arg) {
2    copy_from_user( &mi_dato_kernel, (void __user *)arg, bytes_mi_dato );
3    switch(cmd){
4          case NFPIOC_S2C_DMA:
5            /* ... */
6          case NFPIOC_C2S_DMA:
7            /* ... */
8    }
9    copy_to_user((void __user *)arg, &mi_dato_kernel, bytes_mi_dato);
10 }
```

Cuadro A.10: Código a nivel de driver para la implementación de IOCTL.

Una observación importante es la imposibilidad de acceder a los datos del usuario directamente. Es necesario invocar a las funciones copy_from_user y copy_to_user para la lectura y escritura sobre las direcciones de memoria apuntadas por el usuario.

Manejo de interrupciones en un módulo del kernel

En primer lugar, durante la rutina "probe" hay que habilitar las interrupciones del dispositivo. En este caso, para habilitar interrupciones MSI se siguen las instrucciones del cuadro

A.11. Se indica además, el código que se debe ejecutar para liberar los recursos solicitados en el momento de apagado o expulsión del dispositivo. En el caso de solicitar interrupciones MSIX el procedimiento es similar pero hay que tener en cuenta que es necesario inicializar un vector por cada interrupción solicitada y se realizarán múltiples llamadas a la función request_irq.

```
1  int nfpiface_probe(struct pci_dev *pdev, struct nfp_card *card){
2    pci_enable_msi(pdev);
3    request_irq(pdev->irq, int_handler,  0, DEVICE_NAME, pdev));
4  }
5
6  int nfpiface_remove(struct pci_dev *pdev, struct nfp_card *card){
7    free_irq(pdev->irq, pdev);
8    pci_disable_msi(pdev);
9  }
```

Cuadro A.11: Especificación de la función manejadora.

El segundo parámetro de la función request_irq consiste en un puntero a función, que generalmente encolará el trabajo a realizar para procesarlo en un momento de menor carga (ver cuadro A.12). Para esta acción se utiliza una estructura "workqueue" que se provee entre las herramientas de gestión básicas del sistema Linux y es inicializada en la función "probe".

```
1  irqreturn_t int_handler(int irq, void *dev_id){
2      struct pci_dev *pdev = dev_id;
3      struct nfp_card *card;
4
5      card = (struct nfp_card*)pci_get_drvdata(pdev);
6
7      if ( card->waitingIRQ  ) {
8          queue_work(card->wq, (struct work_struct*)&card->work);
9      } else{
10         return IRQ_HANDLED;
11     }
12     return IRQ_HANDLED;
13 }
14
15 void work_handler(struct work_struct *w){
16     struct nfp_card *card = ((struct my_work_t *)w)->card;
17
18     switch( getDmaState(card,  getEngineOfInterruption(card) )) {
19         case IDLE:
20             break;
21         case BUSY:
22             break;
23         default:
24             printk(KERN_INFO "nfp: There was an error with descriptors")
                   ;
25     }
26 }
```

Cuadro A.12: Implementación de la función manejadora.

B Protocolos de comunicación

Protocolo AXI4-Stream

El protocolo AXI4-Stream es un subconjunto del protocolo AMBA AXI4 [ARM10], diseñado para el flujo de datos a altas velocidades. Las principales características de AXI4-Stream son:

- Facilidad de uso.
  - La configuración más básica del protocolo utiliza únicamente la señal TVALID.
  - Ofrece un conjunto de señales suficientemente amplio como para adaptarse a la mayoría de los casos.
  - Facilidad de conexión entre módulos que implementan el protocolo.
- Flexible.
  - Las reglas predefinidas sobre las señales de usuario siguen funcionando para la mayoría de los diseños.
  - Diferentes topologías se pueden alcanzar por medio de esta interconexión.
  - La unión de distintas señales de datos y la división en recepción está considerada.
  - Existen bytes de posición y bytes nulos de manera predefinida.
- Alto rendimiento. En cada ciclo se pueden transferir datos. Los datos se comunican haciendo un mayor uso del área en lugar del tiempo.

Las señales completas del protocolo se contemplan en la tabla B.1. El protocolo define los siguientes modos de actuación para la transferencia de datos:

- Una transferencia tiene lugar cuando TVALID y TREADY estás activos.
- TKEEP y TSTRB indican si los bytes de la palabra TDATA son de posición o son nulos.
- TLAST indica que un paquete finaliza el flujo de datos actual. Obliga a los datos a ser transferidos al destino si éstos estaban siendo almacenados en un buffer. Puede ser utilizado también para contar los paquetes.
- Un flujo de datos está plenamente identificado por el par TID (identificador) y TDEST (destino).
- El intercalado de transferencias con diferentes flujos está permitido y no se limita a las fronteras delimitadas por la señal TLAST.
- Los datos pueden ser unidos en buses de ancho mayor o separados a buses más estrechos. Ojo, sólo las transferencias con igual TID y TDEST pueden ser concatenadas.
- La señal de reset puede ser alzada asíncronamente. Sin embargo, el cambio posterior (la desactivación) debe ejecutarse después del siguiente flanco ascendente de ACLK.

Finalmente, se describe el efecto de las señales TKEEP y TSTRB sobre los datos en TDATA, tabla B.2.

Nombre señal	Maestro (M), Esclavo(S)	Descripción
ACLK	Clock	En consideración en flanco ascendente.
ARESETn	Reset	Activo a nivel bajo.
TVALID	M	Una transferencia tiene lugar cuando TVALID y TREADY están activos. TVALID indica que el valor en TDATA tiene algún byte válido.
TREADY	S	Una transferencia tiene lugar cuando TVALID y TREADY están activos. TREADY indica que el esclavo está dispuesto a aceptar datos en el presente ciclo de reloj.
TDATA	M	Mensaje primario
TSTRB	M	Marca de byte de posición
TKEEP	M	Marca de byte nulo (byte no habilitado)
TLAST	M	Límite de un paquete
TID	M	Identifica distintos flujos de datos. Generalmente utilizado por utilidades de enrutamiento.
TDEST	M	Ofrece información de enrutado.
TUSER	M	TUSER contiene información lateral, definida por el usuario, que puede ser transmitida junto con el stream principal de datos.

Tabla B.1: Señales del protocolo AXI4-Stream.

TKEEP	TSTRB	Tipo del dato	Descripción
1	1	Byte de dato	El byte asociado contiene información válida que debe ser transferida de la fuente al destino.
1	0	Byte de posición	Indica la posición relativa del byte en el flujo pero no contiene ningún dato relevante.
0	0	Byte nulo	EL byte no contiene información y puede ser eliminado del flujo.
0	1	Reservado	No debe ser usado

Tabla B.2: Señales del protocolo AXI4-Stream: TSTRB y TKEEP.

Fijación del transceptor como elemento de entrada/salida del módulo principal

En la figura 4.9 se mostró como el nivel XGXS se divide en dos componentes, una parte implementada en el transceptor y otra aislada. Para la monitorización de las señales a nivel XAUI desde el módulo top del diseño, basta realizar las correspondientes asociaciones a través del fichero de restricciones en el proyecto. En este caso, el cuadro B.1 muestras las condiciones aplicadas.

```
1  NET "clk_xaui_156" TNM_NET="CLK156";
2  TIMESPEC "TS_CLK156" = PERIOD "CLK156" 156.25 MHz;
3  NET "xaui_ports_i/xaui_dclk" TNM_NET="DCLK";
4  TIMESPEC "TS_DCLK" = PERIOD "DCLK" 50 MHz;
5  NET "xaui_ports_i/xaui_port_0_i/rocketio_wrapper_i/tile1_rxrecclk0_i"
       TNM_NET="CLK156_REC";
6  NET "xaui_ports_i/xaui_port_1_i/rocketio_wrapper_i/tile1_rxrecclk0_i"
       TNM_NET="CLK156_REC";
7  NET "xaui_ports_i/xaui_port_2_i/rocketio_wrapper_i/tile1_rxrecclk0_i"
       TNM_NET="CLK156_REC";
8  NET "xaui_ports_i/xaui_port_3_i/rocketio_wrapper_i/tile1_rxrecclk0_i"
       TNM_NET="CLK156_REC";
9  TIMESPEC "TS_CLK156_REC" = PERIOD "CLK156_REC" 156.25MHz;
10
11 # XAUI 0
12 INST xaui_ports_i/xaui_port_0_i/rocketio_wrapper_i/
       tile0_xaui_v10_4_rocketio_wrapper_i/USE_REVERSE_LANES.gtx_dual_i LOC
       =GTX_DUAL_X1Y6;
13 INST xaui_ports_i/xaui_port_0_i/rocketio_wrapper_i/
       tile1_xaui_v10_4_rocketio_wrapper_i/USE_REVERSE_LANES.gtx_dual_i LOC
       =GTX_DUAL_X1Y7;
14 # XAUI 1
15 INST xaui_ports_i/xaui_port_1_i/rocketio_wrapper_i/
       tile0_xaui_v10_4_rocketio_wrapper_i/USE_REVERSE_LANES.gtx_dual_i LOC
       =GTX_DUAL_X1Y8;
16 INST xaui_ports_i/xaui_port_1_i/rocketio_wrapper_i/
       tile1_xaui_v10_4_rocketio_wrapper_i/USE_REVERSE_LANES.gtx_dual_i LOC
       =GTX_DUAL_X1Y9;
17 # XAUI 2
18 INST xaui_ports_i/xaui_port_2_i/rocketio_wrapper_i/
       tile0_xaui_v10_4_rocketio_wrapper_i/USE_REVERSE_LANES.gtx_dual_i LOC
       =GTX_DUAL_X1Y10;
19 INST xaui_ports_i/xaui_port_2_i/rocketio_wrapper_i/
       tile1_xaui_v10_4_rocketio_wrapper_i/USE_REVERSE_LANES.gtx_dual_i LOC
       =GTX_DUAL_X1Y11;
20 # XAUI 3
21 INST xaui_ports_i/xaui_port_3_i/rocketio_wrapper_i/
       tile0_xaui_v10_4_rocketio_wrapper_i/NO_REVERSE_LANES.gtx_dual_i LOC=
       GTX_DUAL_X0Y11;
22 INST xaui_ports_i/xaui_port_3_i/rocketio_wrapper_i/
       tile1_xaui_v10_4_rocketio_wrapper_i/NO_REVERSE_LANES.gtx_dual_i LOC=
       GTX_DUAL_X0Y10;
23
24 NET "xaui_0_refclk_p"  LOC = "M4" ;
25 NET "xaui_0_refclk_n"  LOC = "M3" ;
26
27 # refclk for Port B
28 #NET "xaui_1_refclk_p"  LOC = "C4" ;
29 #NET "xaui_1_refclk_n"  LOC = "C3" ;
```

```
30
31  # refclk for Port C
32  NET "xaui_2_refclk_p"  LOC = "D16" ;
33  NET "xaui_2_refclk_n"  LOC = "C16" ;
34
35  # refclk for XAUI D
36  NET "xaui_3_refclk_p"  LOC = "D27" ;
37  NET "xaui_3_refclk_n"  LOC = "C27" ;
```

Cuadro B.1: Restricciones asociadas a las señales del protocolo XAUI.